НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

«На правах рукопису»
УДК 004.043

До захисту допущено:
Завідувач кафедри
_____ Ігор ПАРХОМЕЙ
«___»_____2020 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інформаційне забезпечення робототехнічних систем»

зі спеціальності 126 «Інформаційні системи та технології»

на тему: «Оптимізація зображень роботизованої охоронної системи»

Виконав:
студент ІІ курсу, групи ІК-з91МП
Колодій Олександр Валерійович _____

Керівник:
Доцент, к.т.н., доцент
Крилов Євген Володимирович _____

Консультант з нормоконтролю:
доцент, к.т.н., доц.,
Пасько Віктор Петрович _____

Рецензент:
к.т.н., доцент,
Катін Павло Юрійович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Рівень вищої освіти – другий (магістерський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Ігор ПАРХОМЕЙ

«___»_____2020 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Колодію Олександру Валерійовичу

1. Тема дисертації «Оптимізація зображень роботизованої охоронної системи», науковий керівник дисертації доцент, к.т.н., доцент Крилов Є.В., затверджені наказом по університету від « 26 » жовтня 2020р. № 3133-с

2. Термін подання студентом дисертації 14.12.2020

3. Об’єкт дослідження – процес передачі оптимізованих зображень від роботизованої охоронної платформи до серверу для подальшого зберігання.

4. Предмет дослідження – технології, методи та ефективність оптимізації зображень для забезпечення належної роботи системи.

5. Перелік завдань, які потрібно розробити – аналіз аналогічних існуючих систем; аналіз та вибір найкращого формату зображень; вибір технологій для оптимізації зображень та створення системи; проведення експериментального дослідження по оптимізації зображень; розробка системи; розробка програмної реалізації по оптимізації та стисненню зображень.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – архітектура системи, алгоритм оптимізації та надсилання зображень, інтерфейс користувача, таблиця оптимізації, алгоритм отримання та збереження зображень, алгоритм авторизації користувачів

7. Орієнтовний перелік публікацій – 1. Колодій О.В. Розробка системи навчання робототехніці у вигляді веб-додатку // XV MEZINÁRODNÍ VĚDECKO - PRAKTICKÁ KONFERENCE, 22-30 квітня 2019

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Перевірка на співпадіння	доцент Лісовиченко О.І.		
Нормоконтроль	доцент Пасько В.П.		

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Аналіз існуючих охоронних систем, програмно-апаратних рішень	01.10.2020-04.10.2020	
2	Аналіз та тестування форматів зображень та методів їх оптимізації	05.10.2020-09.10.2020	
3	Вибір технологій для створення системи	10.10.2020-11.10.2020	
4	Розробка архітектури системи та методів взаємодії модулів між собою	12.10.2020-18.10.2020	
5	Розробка серверної частини	19.10.2020-25.10.2020	
6	Стиснення розміру зображень та оптимізація процесу їх передачі	26.10.2020-01.11.2020	
7	Розробка клієнтської частини	02.11.2020-08.11.2020	
8	Тестування розробленої системи	09.11.2020-13.11.2020	
9	Оформлення пояснювальної записки	14.11.2020-30.11.2020	
10	Попередній захист		
11	Нормоконтроль		
12	Перевірка на співпадіння		
13	Захист		

Студент Олександр Колодій

Науковий керівник Євгеній Крилов

АНОТАЦІЯ

У даній роботі розглянуто вирішення проблеми оптимізації розміру та процесу передачі зображень.

Насамперед проаналізовано аналогічні системи захисту, визначено їх переваги та недоліки. В результаті отриманих даних визначено необхідні модулі системи, необхідні технології для розробки та методи для подальшої оптимізації зображень та процесу їх передачі.

Результатом виконання даної магістерської дисертації є створена система, що складається з двох серверних частин та однієї клієнтської. Всередині серверної частини робота здійснено стиснення розміру зображень, а також оптимізовано процес передачі файлів від робота до клієнта та сховища.

Для стиснення розміру файлів обрано формат WebP та його алгоритм стиснення, оскільки даний формат має значно менші розміри, у порівнянні з іншими типами зображень. Для оптимізації процесу передачі зображень від рухомої платформи до клієнтів та сховища, всі зображення та кадри відео-потоку переводяться в бінарний вигляд, що зменшує розмір файлів, що передаються.

Ключові слова: стиснення розміру зображень, оптимізація процесу передачі файлів, оптимізація відео-потоку, Node.JS, WebSocket, JavaScript.

Розмір пояснювальної записки – 81 сторінка, 41 ілюстрацій, 28 таблиць, 3 додатки.

ABSTRACT

This paper considers the solution of the problem of optimizing the size and process of image transfer.

First of all, similar protection systems are analyzed, their advantages and disadvantages are identified. As a result of the received data the necessary modules of system, necessary technologies for development and methods for the further optimization of images and process of their transfer are defined.

The result of this master's dissertation is a system consisting of two server parts and one client. Inside the server part of the robot, the image size is compressed, and the process of transferring files from the robot to the client and storage is optimized.

The WebP format and its compression algorithm were chosen to compress the file size, as this format is much smaller than other image types. To optimize the process of transferring images from the mobile platform to clients and storage, all images and frames of the video stream are converted to binary form, which reduces the size of the transferred files.

Keywords: image compression, file transfer optimization, video stream optimization, Node.JS, WebSocket, JavaScript.

Explanatory note size - 81 pages, contain 41 illustrations, 28 tables and 3 applications.

Пояснювальна записка

до магістерської дисертації

на тему: ***Оптимізація зображень роботизованої охоронної системи***

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІЙ УМОВНИХ ПОЗНАЧЕНЬ

HTML – hypertext markup language

CSS – cascading style sheets

JS – JavaScript

Node – Node.JS

HTTP – hypertext transfer protocol

WS – Web Socket

NPM – Node Package Manager

JWT – JSON Web Token

ВСТУП

Підприємства по всьому світу використовують робототехніку для підбору і упаковки замовлень, зварювання металевих деталей, нанесення фарб і герметиків, проведення точних перевірок і багато чого іншого. Тільки в 2018 році було відвантажено рекордну кількість одиниць промислової робототехніки — 381 000 одиниць, що на 30% більше, ніж роком раніше.

Електроніка, автомобілебудування та металургія покладаються на цю технологію для автоматизації виробництва та стимулювання зростання. З його допомогою виробники можуть підвищити продуктивність і продуктивність, зберігаючи при цьому кращу якість і послідовність. Раніше роботи були обмежені в своїх здібностях, вкорінені в одному місці і відокремлені від своїх людських побратимів. Сьогодні, однак, ці машини можуть оцінювати своє оточення і працювати в тандемі з співробітниками.

У той час як роботи вже відіграють помітну роль у логістиці та виробництві, лідери галузі тільки починають усвідомлювати потенціал цієї технології в області заходів безпеки.

Роботи безпеки поступово стають все більш поширеним явищем в торгових центрах, офісах і громадських місцях. Але в той час як ці роботи часто представлені в якості заміни для людей охоронців — дружніх роботів на патрулюванні — вони збирають набагато більше даних, ніж люди могли б, припускаючи, що вони більше схожі на мобільні машини спостереження, ніж звичайні охоронці.

Основною метою цих роботів є збір даних, включаючи номерні знаки, сканування розпізнавання обличчя та наявність поблизу мобільних пристроїв. Це свого роду постійне низькорівневе спостереження, яке може виконувати тільки машина.

Роботи безпеки – це, по суті, додаткові пристрої, покликані компенсувати нестачу персоналу, виявляти неприємності і викликати поліцію. Але в епоху, коли автоматизовані системи замінюють людей у все більшій кількості областей, цілком імовірно, що вони поступово візьмуть на себе більш помітну роль, спираючись на свої навички спостереження.

На сьогодні відомо безліч різних систем захисту, але досі основною залишається система відеоспостереження за допомогою спеціальних відеокамер закріплених на стінах або на стелі, які передають та зберігають інформацію на захищеному сервері. Власник такої системи має доступ до серверу та має можливість переглядати збережену інформацію або слідкувати за приміщенням дистанційно онлайн.

Для оптимальної роботи відеоспостереження необхідно мати камеру з хорошою якістю зйомки, належну швидкість інтернету для передачі даних до серверу та велику кількість пам’яті, щоб зберігати відеодані за тривалий період часу. Щоб підвищити швидкість передачі даних та зменшити навантаження на сервер необхідною умовою є оптимізація зображення без великої втрати якості та можливе зменшення кількості кадрів за секунду.

Одним з перспективних підходів для забезпечення захисту будинку є роботизована керована платформа, яка веде фотозйомку. Дана система надає можливість користувачеві дистанційно керувати напрямом робота, спостерігати за поточним станом приміщення або переглядати історію зйомок. Важливим для такої системи є оптимізація розміру зображень, які передаються на сервер.

Таким чином у роботі розв’язується актуальна задача захисту приміщень за допомогою роботизованої рухомої платформи, яка буде вести фотозйомку, здійснювати оптимізацію зображень, відправляти оптимізовані зображення на сервер та зберігати їх там.

Метою дослідження є підвищення ефективності процесу передачі зображень від камери роботизованої рухомої платформи до серверу для підвищення швидкодії та зменшення навантаження на сервер, за рахунок оптимізації розміру зображень та можливості зменшення кількості зроблених кадрів за секунду.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Об’єкт та предмет дослідження

Дослідження проводиться в області процесу передачі та оптимізації зображень від камери до серверу. Останнім часом якість і відповідно розмір відеоматеріалів та зображень доволі швидко зросла, тому постає завдання в їх оптимізації для надання можливості зберігати та передавати більший обсяг даних за одиницю часу.

Об’єкт дослідження – процес передачі оптимізованих зображень від роботизованої охоронної платформи до серверу для подальшого зберігання.

Предмет дослідження – технології, методи та ефективність оптимізації зображень для забезпечення належної роботи системи.

1.2 Огляд існуючих функціональних можливостей систем захисту

Наразі, більшість сучасних систем захисту мають велику кількість функціональних можливостей і значно спрощують роботу користувача з ними.

Датчики руху, які дозволяють записувати відео не постійно – навантажуючи сервер та заповнюючи локальний або хмарний накопичувач – а тільки у момент, поки щось не переміститься в кадрі. Також, є можливість отримувати сповіщення на свій комп'ютер або телефон, якщо щось рухається, коли це не повинно відбуватися. Приклад сповіщення зображено на рис. 1.1

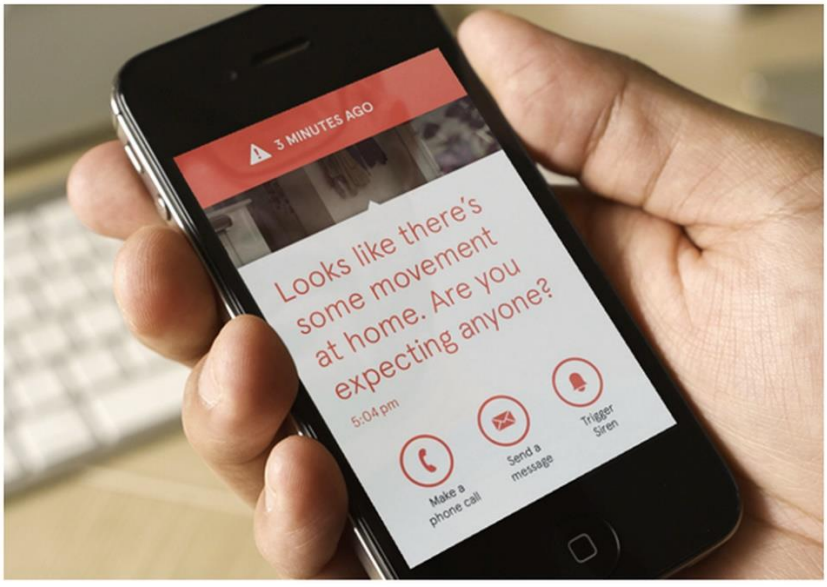


Рисунок 1.1 – Сповіщення про рух у приміщенні

Двостороннє аудіо – динаміки на камері відеоспостереження, що забезпечують двосторонній зв'язок між людиною, що спостерігає за відеопотоком, і людиною, що знаходиться перед камерою.

Камери, з хорошою роздільною здатністю та високою якістю зображення. Протягом багатьох років можливості камер безпеки значно відставали від дозволу запису, дозволених на звичайних відеокамерах, виробляючи переривчасте, піксельне відео з повільною частотою оновлення. Більшість сучасних камер відеоспостереження пропонують дозвіл не менше 720p, а багато з них навіть пропонують до 1080p для запису і потокової передачі.

Хоча відео високої чіткості безпеки звучить як ідеальний спосіб стежити за будинком, воно створює свою власну проблему: потокове передавання одного або декількох каналів 1080p може втрачати пропускну здатність домашнього інтернету.

Основною функціональною можливістю системи захисту є хмарне або локальне сховище. Камера стеження, яка не зберігає і не записує те, що вона захоплює, не дуже ефективна, особливо якщо необхідно переслідувати зловмисника. Ось чому більшість камер відеоспостереження пропонують або локальне, або хмарне сховище (а іноді, хоча і рідко, і те, і інше).

Локальне сховище – це вірний спосіб зафіксувати все, що відбувається. Відеопотік зберігається безпосередньо на сусідньому накопичувачі у вигляді відео, але є одна проблема. Якщо записувати все (а не тільки при виявленні руху), то скоро диск заповниться і більше не зможе зберігати відзнятий матеріал, поки жорсткий диск не буде очищено.

З хмарним сховищем можна обійти цю головоломку, але є ще більше застережень. Варіанти зберігання даних зазвичай стягують щомісячну або річну плату і обмежують кількість відеоматеріалів, які можна зберігати, або кількома днями, або кількістю відеокліпів, збережених в акаунті.

На рис. 1.2 зображено доступ до сховища через мобільний додаток.

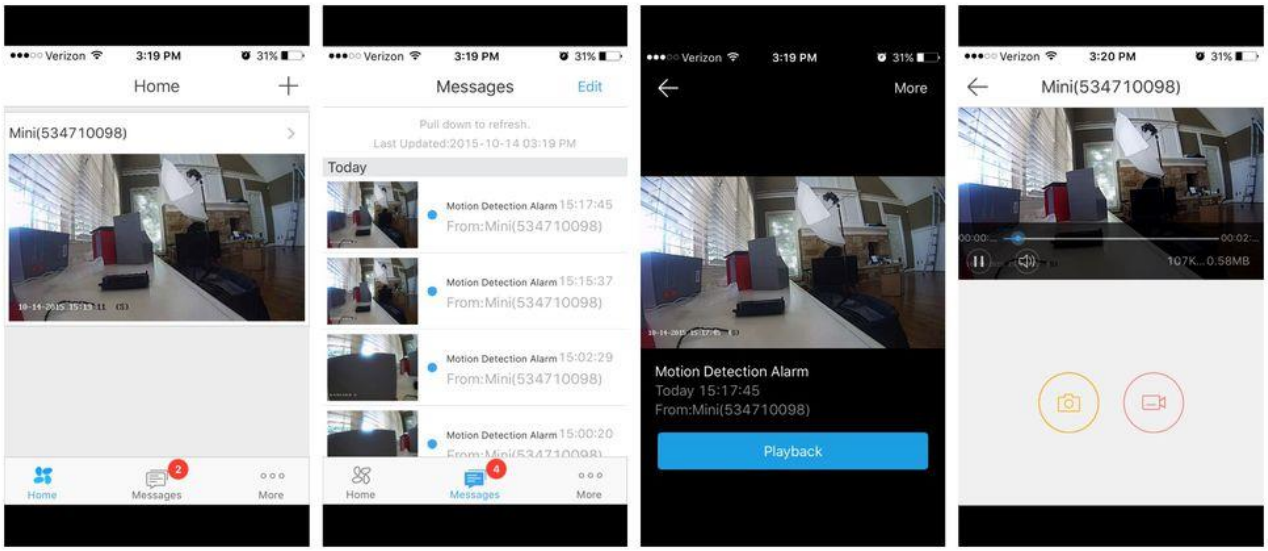


Рисунок 1.2 – Мобільний додаток для доступу до сховища

1.3 Аналіз існуючих систем захисту приміщень

Хоча сучасні камери для захисту приміщень мають велику кількість функціональних можливостей, досі основною проблемою залишається збереження обмеженої кількості даних на внутрішньому накопичувачі пристрою, а для розширення – необхідно використовувати хмарні сервіси, за які потрібно буде платити щомісячно або щорічно. А враховуючи роздільну здатність сучасних камер, накопичувач буде заповнюватись неймовірно швидко.

Іншою проблемою є обмежена можливість доступу для віддаленого перегляду за тим, що відбувається в даний час у приміщенні, або для перегляду історії, оскільки для доступу існують певні програми, якими можуть користуватись тільки користувачі певних мобільних пристроїв.

Розглянемо камеру для захисту приміщень, Arlo Pro4, зображену на рис. 1.3.

Рисунок 1.3 – Камера Arlo Pro4

За допомогою спеціального додатку, Arlo дозволяє отримувати доступ до камер і управляти ними за допомогою смартфона. Система виявлення руху дозволяє камерам економити заряд батареї та зменшувати завантаження сховища, записуючи тільки тоді, коли є рух перед камерою.

Певним недоліком системи є необхідність купівлі обов’язкової базової станції. Але разом з цим, вони покращують час автономної роботи в порівнянні з камерами безпеки, які використовують тільки Wi-Fi. Вони також забезпечують локальне зберігання відео, підключаючи запам'ятовуючий пристрій до концентратора. Базова станція – це найкращий спосіб записувати локальне відео без хмарної підписки, що в довгостроковій перспективі економить більше грошей.

Камера надає вихідне зображення із роздільною якістю 1440 пікселів (2К) та кутом огляду в 160 градусів. Однак розмір вихідного файлу є досить великим, тому накопичувач доволі швидко заповнюється і користувачу необхідно видалити не потрібні файли з накопичувача.

Користувач має можливість переглядати відео в реальному часі в додатку Arlo – який необхідно завантажити на свій мобільний пристрій – в будь-який час без підписки, але також надає додаткові функції і більш високу якість зберігання відео тільки з його платними планами Arlo Smart.

Розглянемо ще одну камеру призначену для захисту приміщень, Reolink Argus 2, яка зображена на рис. 1.4.

Рисунок 1.4 – Камера Reolink Argus 2

Reolink – це IP-камера, тобто користувач може мати доступ до прямої відео трансляції з будь-якого місця за допомогою програми Reolink, яку необхідно завантажити на смартфон. Камера має вбудований датчик руху, який економить час автономної роботи, записуючи на карту micro SD тільки тоді, коли датчик виявляє

певний рух, а не працює безперервно. Мобільний додаток дозволяє налаштувати сповіщення, або змінити роздільну здатність зображення.

До переваг можна віднести можливість зберігати відео у хмарні сервіси безкоштовно та без необхідності у зовнішньому накопичувачі. Якщо ж інтернет мережа не є доступною, у такому разі необхідно купити зовнішній накопичувач, щоб мати можливість зберігати відзнятий відеоматеріал.

Також за допомогою мобільного додатку є змога змінювати налаштування якості зображення для різних переваг, таких як збільшення роздільної здатності для більшої чіткості або зниження його для кращого часу автономної роботи і кращої продуктивності при більш низькій швидкості з'єднання. За допомогою програми також можна знімати відео і фотографії з прямої трансляції і зберігати їх на свій телефон, навіть без карти microSD, а також надавати іншим членам сім'ї доступ до прямої трансляції за допомогою QR-коду.

Великим недоліком є те, що при максимальній швидкості відео всього 15 кадрів в секунду, відео потік Reolink Argus 2 набагато більш переривчастий, ніж у інших наших камер безпеки. Існує також приблизно 2,5-секундна затримка, перш ніж камера викличе сигнал тривоги про рух, тому камера не встигне зняти необхідний матеріал.

1.4 Постановка задачі

Метою магістерської дисертації є підвищення ефективності процесу передачі зображень від камери роботизованої рухомої платформи до серверу для підвищення швидкодії та зменшення навантаження на сервер, за рахунок оптимізації розміру зображень та можливості зменшення кількості зроблених кадрів за секунду.

Для досягнення поставленої мети в роботі визначені наступні завдання:

- Проаналізувати існуючі функціональні можливості та аналоги систем захисту приміщень, виявити їх переваги та недоліки;
- На основі аналізу сучасного стану різних типів форматів зображень та можливості їх оптимізації, обрати найкращий тип для подальшого стиснення розміру зображення;

- Провести експериментальне дослідження швидкодії різних методів оптимізації зображень та обрати той метод, в результаті якого, отримане зображення буде одним з найменших по розміру та нас буде задовольняти його якість;
- Розробити програмну реалізацію по оптимізації зображень, які надходять з відеокамери та передача оптимізованих файлів до серверу;
- Розробка веб додатку для надання можливості користувачеві переглядати файли, збережені на сервері.

Висновок до розділу

В даному розділі розглянуто основні функціональні можливості сучасних систем захисту приміщень, а також проведено аналіз існуючих найпопулярніших систем, виявлено основні їх переваги та недоліки. Першою розглянутою системою була камера Arlo Pro4, недоліками якої була необхідність у придбанні базової станції, на яку здійснюється запис файлів, а для розширення і доступу до хмарного сховища необхідна щомісячна або річна підписка. Також, не дуже зручним є необхідність завантажувати мобільний додаток на свій смартфон, щоб отримати доступ до віддаленого перегляду за камерою. Критичним є те, що за допомогою комп'ютера, можливості підключитись віддалено не існує. До переваг даної системи можна віднести дуже хорошу якість зйомки та велику кількість функціональних можливостей, таких як система виявлення руху або нічна зйомка. Інша розглянута система має досить схожі функціональні можливості, але на відміну від першої – файли можна зберігати у хмарне сховище безкоштовно, без необхідності у зовнішніх накопичувачах інформації, що є дуже хорошим плюсом. В даній системі так само необхідно завантажувати мобільний додаток, для доступу до камери, хоча функціональних можливостей у даному додатку значно більше, оскільки користувач має змогу не тільки переглядати файли і дивитись, що відбувається в приміщенні в даний момент, а й налаштувати якість зйомки. Основним недоліком системи є обмеженість відео в 15 кадрів за секунду та велика переривчастість відео потоку.

Врахувавши усі проаналізовані переваги та недоліки обох систем визначено мету, предмет та об’єкт дослідження. Також визначено основні задачі, які необхідно виконати для досягнення зазначеної мети.

РОЗДІЛ 2. ВИБІР ФОРМАТУ ЗОБРАЖЕННЯ ТА РІВНЯ ЇХ СТИСНЕННЯ

2.1 Аналіз форматів зображень

Існує дуже велика кількість унікальних за своєю структурою та призначенням різних форматів зображень. Одні, підходять для чорно-білих зображень та дозволяють масштабувати його, без зміни якості та розміру зображень. Інші, призначені для кольорових фотографій, які містять дуже велику палітру кольорів. Розглянемо приклад векторного (який знаходиться зліва на рисунку) та растрового (який знаходиться справа на рисунку) типів зображень, які продемонстровані на рис. 2.1.

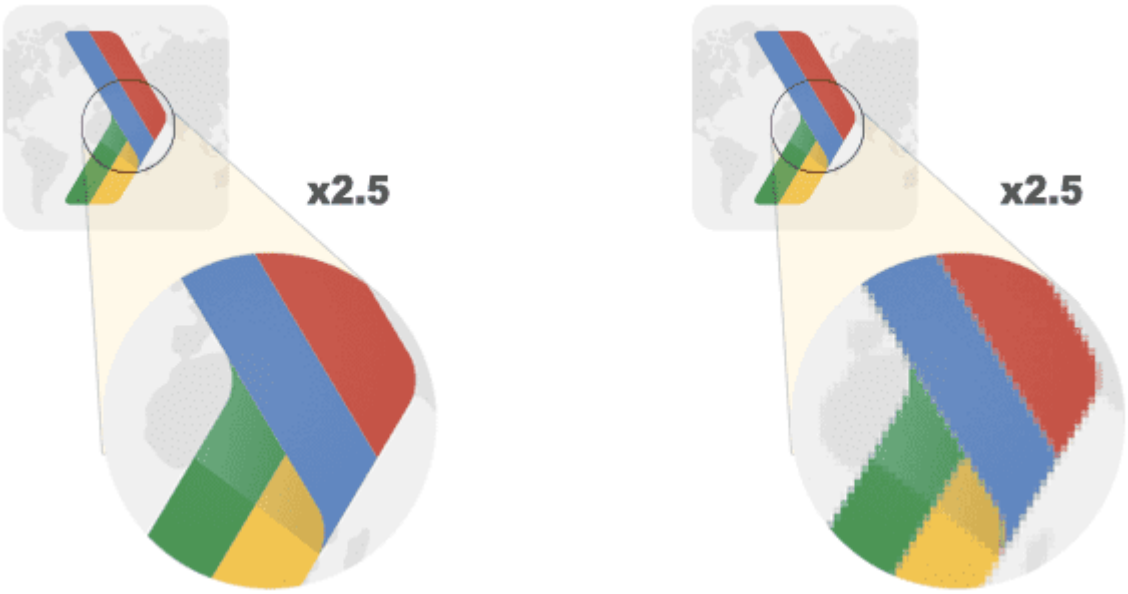


Рисунок 2.1 – Збільшене векторне та растрове зображення

З рисунку вище можемо зробити висновок:

- Векторна графіка використовує лінії, точки та багатокутники для подання зображення;
- Растрова графіка представляє зображення шляхом кодування окремих значень кожного пікселя в прямокутній сітці.

У кожного формату є свій набір плюсів і мінусів. Векторні формати ідеально підходять для зображень, що складаються з простих геометричних фігур, таких як логотипи, текст або значки. Вони забезпечують чіткі результати при будь-якій роздільній здатності та масштабуванні, що робить їх ідеальним форматом для

екранів з високою роздільною здатністю і активів, які повинні відображатися в різних розмірах.

Однак векторні формати не підходять, коли сцена складна (наприклад, фотографія): обсяг розмітки SVG для опису всіх фігур може бути непомірно високим, і результат все одно може не виглядати "фото реалістичним". У цьому випадку необхідно використовувати растровий формат зображення, такий як PNG, JPEG або WebP.

Растрові зображення не володіють такими ж хорошими властивостями, як роздільна здатність або масштабування незалежно – при масштабуванні растрового зображення можна побачити нерівну і розмиту графіку. В результаті може знадобитися зберегти кілька версій растрового зображення в різній роздільній здатності, щоб забезпечити оптимальний досвід для різних користувачів.

Існує два різних типи пікселів: CSS-пікселі і пікселі пристроїв. Один піксель CSS може відповідати безпосередньо одному пікселю пристрою або може підтримуватися декількома пікселями пристрою. Чим більше пікселів пристрою, тим дрібніше деталізація відображуваного контенту на екрані. Приклад різниці продемонстровано на рис. 2.2.

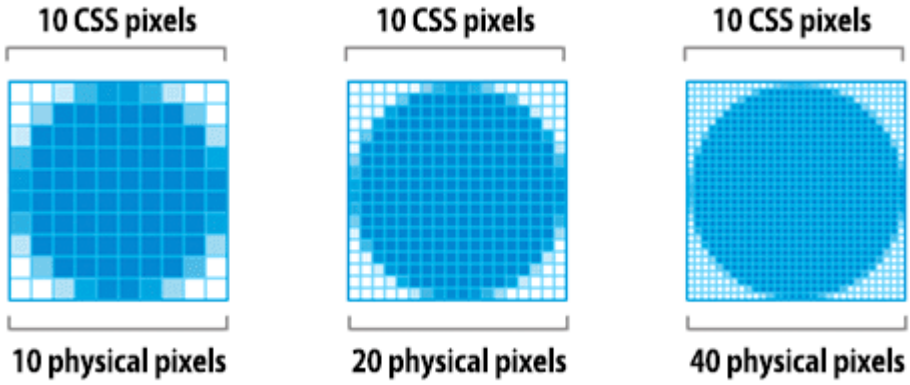


Рисунок 2.2 – Різниця між CSS-пікселями і пікселями пристрою.

Екрани з високою роздільною здатністю dpi (HiDPI) дають прекрасні результати, але є один очевидний компроміс: графічні зображення вимагають більшої деталізації, щоб скористатися більш високою кількістю пікселів пристрою. Хороша новина полягає в тому, що векторні зображення ідеально підходять для цього завдання, оскільки вони можуть бути візуалізовані в будь-якій роздільній здатності з різкими результатами. Ймовірно, будуть більш високі витрати на

обробку, щоб візуалізувати більш дрібні деталі, але базове зображення залишається тим же самим і не залежить від роздільної здатності.

З іншого боку, растрові зображення є набагато більшою проблемою, оскільки вони кодують дані зображення на основі кожного пікселя. Отже, чим більше число пікселів, тим більший розмір файлу растрового зображення. Як приклад розглянемо різницю між фото розміром 100x100 (CSS) пікселів (табл. 2.1).

Таблиця 2.1 – Розмір растрового зображення

Роздільна здатність екрана	Загальна кількість пікселів	Розмір нестисненого файлу (4 байти на піксель)
1x	100 x 100 = 10,000	40 000 байт
2x	100 x 100 x 4 = 40,000	160,000 байт
3x	100 x 100 x 9 = 90,000	360,000 байт

Коли ми подвоюємо роздільну здатність фізичного екрану, загальна кількість пікселів збільшується в чотири рази: подвоюємо кількість горизонтальних пікселів, вдвічі більше вертикальних пікселів. Отже, екран "2x" не просто подвоюється, а в чотири рази збільшує кількість необхідних пікселів!

На практиці, екрани з високою роздільною здатністю дозволяють отримувати прекрасні зображення, що може бути чудовою функцією продукту. Однак для екранів з високою роздільною здатністю також потрібні зображення з високою роздільною здатністю, тому:

- Необхідно віддавати перевагу векторним зображенням, коли це можливо, оскільки вони не залежать від роздільної здатності і завжди дають різкі результати;
- Якщо потрібне растрове зображення, необхідно створювати відповідні зображення до різної роздільної здатності.

Оскільки наша задача оптимізувати розмір зображень, а саме – фотографії, то растровий формат для нас підходить більше. Необхідною умовою залишилось обрати необхідний формат серед PNG, JPEG та WebP.

2.2 Особливості різних форматів растрових зображень

На додаток до різних алгоритмів стиснення з втратами і без втрат, різні формати зображень підтримують різні функції, такі як анімація і прозорість

(Альфа) каналів. В результаті вибір "правильного формату" для конкретного зображення являє собою комбінацію бажаних візуальних результатів і функціональних вимог (табл. 2.2).

Таблиця 2.2 – Формати та візуальні ефекти

Формат	Прозорість	Анімація	Підтримка браузерів
PNG	Так	Ні	Усі
JPEG	Ні	Ні	Усі
WebP	Так	Так	Тільки сучасні

Існує два універсально підтримуваних формату растрових зображень: PNG і JPEG. На додаток до цих форматів сучасні браузери підтримують більш новий формат WebP, який забезпечує краще загальне стиснення і більше можливостей. Формат WebP, як правило, забезпечує краще стиснення, ніж старі формати, і його слід використовувати там, де це можливо. Його можна використовувати разом з іншим форматом зображення як запасний варіант.

2.2.1 JPEG формат

JPEG (він же JPG) – це формат зображень, який використовує стиснення з втратами і не підтримує прозорість. Дозволяє налаштовувати рівень якості зберігання зображення – при його зниженні видаляються деталі і додаються шуми на зображення, проте розмір стає більш компактним. JPG в залежності від налаштувань може забезпечити стиснення як 2: 1, так і 100: 1 - але якість прямо пропорційна коефіцієнту стиснення. Назва формату – абревіатура від Joint Photographic Experts Group.

JPEG підтримує колірні палітру 24-bit RGB і CMYK, а також 8-bit Grayscale. CMYK і Grayscale використовуються досить рідко і їх підтримка викликає нарікання.

Також JPEG має інтегровану підтримку EXIF, що дозволяє зберігати метадані, наприклад: виробник і модель використаної камери, яка використовується для зйомки витримка, діафрагма і світлочутливість, дозвіл кадру, налаштування балансу білого, фокусна відстань, використання спалаху, розмір матриці, дата і час зйомки, географічні координати і адресу місця зйомки.

Використовувані розширення для файлів - .jpg and .jpeg (працюють ідентично).

2.2.2 PNG формат

PNG 24 – це формат зображень, який працює з кольоровими зображеннями, використовує стиснення без втрат і дозволяє зберігати прозорість. Налаштувати якість збереження в PNG 24 неможливо, однак, можна адаптувати зберігання зображення для досягнення мінімального розміру файлу: для цього можна знизити кількість кольорів в зображенні. Назва формату – акронім від Portable Network Graphics.

Існує також формат PNG 8 – він більш компактний, ніж PNG 24, але можна застосувати тільки для зображень з дуже обмежених кількістю кольорів: 256 – це максимум. У разі використання PNG 8 для зображень з великою кількістю кольорів, стиснення буде з втратами і з ефектом пастеризації.

PNG до 2017 року не підтримував EXIF, але потім його підтримка була реалізована в стандарті. У фотографії PNG використовується рідко – для компактного зберігання файлів більше підходить JPEG, а для професійної роботи краще підходять RAW-формати DNG або TIFF.

PNG 24 і PNG 8 використовують розширення для файлів .png, використовувана бітність записується в метадані файлу і по розширенню не визначається.

2.2.3 WebP формат

WebP – це новий формат зображень, створений у 2010 році і в даний час розробляється Google на основі технології, придбаної внаслідок придбання On2 Technologies. Він забезпечує чудове стиснення без втрат та втрат зображень, зберігаючи при цьому незмінну якість. Ступінь стиснення з втратами регулюється, що означає, що ми можемо визначити компроміс між розміром та якістю.

Існує два види зображень WebP. Один з них відомий просто як WebP і використовує стиснення з втратами. Для даного формату є можливість вибрати ступінь стиснення. Інша версія відома як WebP Lossless і більше схожа на PNG.

Такий формат має більший розмір файлу, але не втрачає деталізацію. Обидва типи форматів WebP створюють зображення, яке набагато менше, ніж їх аналоги JPEG і PNG.

Перевагами формату WebP є менший розмір файлу з рівноцінною якістю порівняно з PNG та JPEG. Формат WebP без втрат може стискати зображення на 26% більше порівняно з PNG, тоді як зображення з втратами WebP на 25-35% менші за розміром при еквівалентній якості у порівнянні з JPG.

Недоліками даного формату є погана сумісність. Хоча й більшість сучасних браузерів вже підтримують формат WebP, Internet Explorer не підтримує даний формат взагалі, а Safari має тільки часткову сумісність. За винятком браузера, загальне програмне забезпечення графічного редактора не може відкривати або редагувати зображення WebP.

2.2.4 MJPEG формат

MJPEG – це неофіційний стандарт кодування відео, де кожний окремий кадр послідовно незалежно закодований алгоритмом JPEG. Наразі, даний формат найбільш поширено використовується в цифрових камерах та системах зовнішнього спостереження, оскільки надає стоп-кадри високої якості. У порівнянні з алгоритмом MPEG-4, даний алгоритм вимагає значно менше обчислювальних ресурсів, його апаратна реалізація значно простіша та відповідно дешевша, а також у ньому простіше реалізовується функція прискореного відтворення, як вперед, так і назад.

Формат MJPEG зручно використовувати для редагування відео, оскільки зв'язок між сусідніми кадрами відсутній і можна вносити зміни в кадр, не побоюючись, що це вплине на сусідні кадри. На відміну від групи стандартів MPEG, в алгоритмах MJPEG не передбачено усунення часової надмірності. Тому ступінь стиснення у алгоритмів MJPEG є нижчим (близько 5-20) за однакової якості зображення.

Одним з основних недоліків формату MJPEG є відсутність єдиного документа, який би визначав специфікацію формату для всіх. Тому можна зустріти

реалізації MJPEG від різних розробників і виробників ПЗ, що не сумісні один з одним.

2.3 Рівні стиснення зображень

Зображення часто становлять більшість завантажених байтів на веб-сторінці, а також часто займають значну кількість візуального простору. Як результат, оптимізація зображень часто може принести деякі найбільші заощадження байтів та покращення продуктивності веб-сайту: чим менше байт повинен завантажити браузер, тим менша конкуренція за пропускну здатність клієнта і швидше браузер може завантажити та зробити корисним вміст на екрані.

Оптимізація зображення – це як мистецтво, так і наука: мистецтво, оскільки немає однозначної відповіді на те, як найкраще стиснути окреме зображення, і наука, оскільки існує безліч добре розроблених методів та алгоритмів, які можуть значно зменшити розмір зображення. Пошук оптимальних налаштувань для вихідного зображення вимагає ретельного аналізу за багатьма вимірами: можливості форматування, вміст закодованих даних, якість, розміри пікселів тощо.

Растрове зображення – це просто двовимірна сітка окремих "пікселів" – наприклад, зображення розміром 100x100 пікселів є послідовністю 10000 пікселів. У свою чергу, кожен піксель зберігає значення " RGBA ": (R) червоний канал, (G) зелений канал, (B) синій канал та (A) альфа-канал (прозорість).

Внутрішньо браузер виділяє 256 значень (відтінків) для кожного каналу, що перетворюється на 8 бітів на канал (2 ^ 8 = 256) та 4 байти на піксель (4 канали x 8 біт = 32 біти = 4 байти). Як результат, якщо ми знаємо розміри сітки, ми можемо легко розрахувати розмір файлу:

- Зображення 100x100 пікселів складається з 10000 пікселів;
- 10000 пікселів x 4 байти = 40000 байт;
- 40000 байт / 1024 = 39 КБ.

Окрім того, незалежно від формату зображення, що використовується для передачі даних із сервера на клієнт, коли зображення декодується браузером, кожен піксель завжди займає 4 байти пам'яті. Це може бути важливим обмеженням для

великих зображень та пристроїв, які не мають багато вільної пам'яті – наприклад, мобільних пристроїв низького класу.

На табл. 2.3 наведено залежність розміру файлу в залежності від розміру зображення у пікселях.

Таблиця 2.3 – Залежність розміру файлу від розміру зображення

Розміри	Пікселі	Розмір файлу
100 x 100	10 000	39 КБ
200 x 200	40 000	156 КБ
300 x 300	90 000	351 КБ
500 x 500	250 000	977 КБ
800 x 800	640 000	2500 КБ

39 Кб для зображення розміром 100x100 пікселів може здатися не великою справою, але розмір файлу швидко збільшується для більших зображень і робить завантаження графічних ресурсів повільним.

Одна проста стратегія полягає в зменшенні "бітової глибини" зображення з 8 біт на канал до меншої кольорової палітри: 8 біт на канал дають нам 256 значень на канал і 16 777 216 (256 ^ 3) кольорів в цілому. Якщо зменшити палітру до 256 кольорів, тоді знадобиться всього 8 біт для каналів RGB і відразу ж можна заощадити два байти на піксель – це 50% економія стиснення в порівнянні з вихідним форматом 4 байта на піксель.

На рис. 2.3 зліва направо йдуть зображення: 32-бітний (16M кольорів), 7-бітний (128 кольорів), 5-бітний (32 кольори).

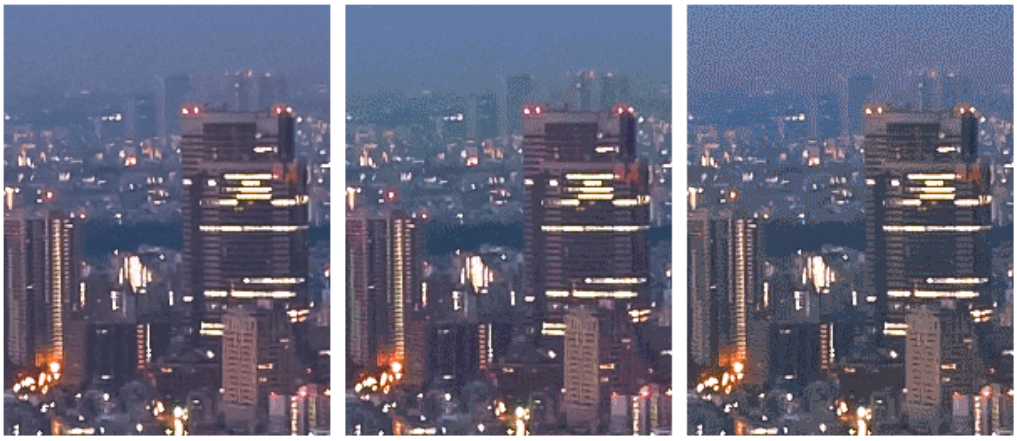


Рисунок 2.3 – Зменшення бітової глибини зображення

Складні сцени з поступовими колірними переходами (наприклад, градієнти або небо) вимагають великих колірних палітр, щоб уникнути візуальних артефактів, таких як піксельне небо в 5-бітному активі. З іншого боку, якщо зображення використовує лише кілька кольорів, то велика палітра просто витрачає дорогоцінні біти.

Після оптимізації даних, що зберігаються в окремих точках, можна спробувати подивитися на сусідні пікселі: оскільки, багато зображень, і особливо фотографії, мають багато сусідніх пікселів з однаковими кольорами – наприклад, небо, повторювані текстури і т. д. Використовуючи цю інформацію в своїх інтересах, компресор може застосувати Дельта-кодування, де замість зберігання окремих значень для кожного пікселя, можна зберігати різницю між сусідніми пікселями: якщо сусідні пікселі однакові, то дельта дорівнює "нулю", тому необхідно зберегти тільки один біт.

Людське око має різний рівень чутливості до різних кольорів: тому можна оптимізувати своє кодування кольорів, зменшуючи або збільшуючи палітру цих кольорів. "Сусідні" пікселі утворюють двовимірну сітку. Це означає, що кожен піксель має кілька сусідів: використаємо цей факт для подальшого поліпшення Дельта-кодування. Замість того щоб розглядати тільки найближчих сусідів для кожного пікселя, потрібно подивитися на більші блоки сусідніх пікселів і кодувати різні блоки з різними налаштуваннями.

Для деяких типів даних, таких як вихідний код сторінки або виконуваний файл, дуже важливо, щоб компресор не змінював і не втрачав вихідну інформацію: один відсутній або неправильний біт даних може повністю змінити значення вмісту файлу або, що ще гірше, повністю його зруйнувати. Для деяких інших типів даних, таких як зображення, аудіо та відео, може бути цілком прийнято надати "приблизне" уявлення вихідних даних.

Насправді, через те, як працює око, ми часто можемо відмовитися від деякої інформації про кожен піксель, щоб зменшити розмір файлу зображення-наприклад, наші очі мають різну чутливість до різних кольорів, що означає, що ми можемо використовувати менше бітів для кодування деяких кольорів. В результаті Типовий конвеєр оптимізації зображень складається з двох етапів високого рівня:

- Зображення обробляється за допомогою фільтра з втратами, який усуває деякі піксельні дані;
- Зображення обробляється за допомогою фільтра без втрат, який стискає піксельні дані.

Перший крок необов'язковий, і точний алгоритм буде залежати від конкретного формату зображення, але важливо розуміти, що будь-яке зображення може пройти етап стиснення з втратами, щоб зменшити його розмір. Насправді різниця між різними форматами зображень, такими як GIF, PNG, JPEG та іншими, полягає в поєднанні конкретних алгоритмів, які вони використовують (або опускають) при застосуванні кроків з втратами і без втрат.

Не існує оптимальної конфігурації для оптимізації з втратами і без втрат. Але все залежить залежить від вмісту зображення та власних критеріїв, таких як компроміс між розміром файлу та артефактами, введеними стисненням із втратами: у деяких випадках ви можете пропустити оптимізацію з втратами, щоб передати складні деталі в повній точності. В інших випадках можна застосувати агресивну оптимізацію з втратами, щоб зменшити розмір файлу ресурсу зображення. Саме тут обрати власне судження і контекст, оскільки немає ніякої універсальної установки.

Як практичний приклад, при використанні формату з втратами, такого як JPEG, компресор зазвичай надає можливість налаштувати параметр "якість", який зазвичай являє собою число від 1 до 100, що управляє внутрішньою роботою конкретної колекції алгоритмів з втратами і без втрат. Для досягнення найкращих результатів необхідно експериментувати з різними налаштуваннями якості зображень.

Рівні якості для різних форматів зображень безпосередньо не порівнянні через різницю в алгоритмах, що використовуються для кодування зображення: якість 90 JPEG дасть зовсім інші результати, ніж якість 90 WebP. Насправді навіть рівні якості для одного і того ж формату зображення можуть давати помітно різні результати на основі реалізації компресора.

Висновок до розділу

В даному розділі розглянуто основні формати зображень, а саме: векторний та растровий. Векторний формат підходить для зображень, що складаються з простих геометричних фігур, таких як логотипи, текст або значки. Даний формат не втрачає якості при збільшенні роздільної здатності екрану. Однак векторні формати не підходять для різного роду фотографій, де міститься велика кількість різних об’єктів та велика кількість швидких переходів різних кольорів. Оскільки основною задачею є оптимізація саме фотографій, то було обрано растровий формат. Даний формат у свою чергу містить 3 типи зображень: PNG, JPEG та WebP. У кожного з них є свої переваги та недоліки. Зображення формату PNG мають дуже велику роздільну здатність та високу якість, але у свою чергу є дуже великі за розмірами та не мають можливості до стиснення. JPEG – хоча й не має такої ж хорошої якості, але використовує власні алгоритми для стиснення, тому кінцевий користувач має змогу налаштувати цей показник під свої потреби. Останній, WebP формат, найкраще підходить для реалізації на веб-сторінках, оскільки створений саме для цього та має новітні алгоритми по стисненню зображення, в результаті чого зображення на 25-35% менші за розміром при еквівалентній якості у порівнянні з JPG.

РОЗДІЛ 3. ВИБРІ ТЕХНОЛОГІЙ ДЛЯ ПРОЦЕСУ ОПТИМІЗАЦІЇ ЗОБРАЖЕНЬ, РОЗРОБКИ СЕРВЕРУ ТА ВЕБ ДОДАТКУ

До основних задач даної дисертації можна віднести процес оптимізації розміру зображень, передача їх від роботизованої платформи до сховища та представлення користувачеві збережених файлів. Однією із складових виступає серверна частина, створена на платформі робота. Сервер отримує зображення з камери, стискає файли, частину файлів передає кінцевому користувачеві у вигляді відео потоку, а частину файлів зберігає у сховищі. Другою складовою є сховище, на якому також необхідно створити сервер для отримання, зберігання та представлення зображень користувачеві. Для створення серверної частини доцільно обрати програмну платформу Node.JS. Щоб отримати необхідні модулі, потрібно встановити менеджер пакунків NPM. В якості бази даних, для зберігання списку користувачів, які мають доступ до серверів, обрано MongoDB. Для реалізації клієнтської частини слід обрати мову розмітки документів HTML, а для наділення її певними візуальними властивостями – використаємо каскадні таблиці стилів CSS. Також необхідно здійснити отримання зображень від серверу та представлення їх у веб-додатку за допомогою мови програмування JavaScript та використання протоколу WebSocket. Остання складова – стандарт JWT, який використовується для верифікації користувачів.

Розглянемо описані технології більш детально.

3.1 Node.JS та npm

Основою для створення веб-серверу є програмна платформа NodeJS.

NodeJS – програмна платформа з відкритим кодом, яка заснована на базі Google рушія V8, який конвертує мову програмування JavaScript в машинний код для підвищення швидкодії. Основним призначенням платформи є виконання високопродуктивних мережевих застосунків, які написані на JS. Вона добавляє можливість JS взаємодіяти з пристроями введення-виведення за допомогою API.

API – опис можливих варіацій, за допомогою яких одна комп’ютерна програма може взаємодіяти з іншою. Зазвичай, є частиною певного інтернет-протоколу, такого як HTTP.

Платформа не тільки працює із серверними додатками для виконання веб-запитів, а й використовується для створення серверних та клієнтських програм. В основі платформи використана асинхронна модель з неблокуючим введенням-виводом, що дозволяє обробляти велику кількість різних запитів одночасно.

Також, важливо враховувати, що в основі лежить подійно-орієнтоване програмування. Тому, замість виконання певного запиту з очікуванням завершення його роботи та наступною обробкою результату, в платформі використовується принцип асинхронного виконання. Розглянемо приклад синхронного виконання команди:

```
var result = db.query ("select .. ");
```

розглянемо приклад асинхронного виконання команди:

```
db.query ("select .. ", function (result) { /* обробка результату */ });
```

Отже, при синхронному виконанні команди, спершу, ми очікуємо на результат виконання команди, а вже після цього можемо переходити до виконання наступного блоку коду. При асинхронному виконанні, управління відразу перейде до наступного блоку коду, не затримуючи виконання наступних команд. Для отримання ж результату та його обробки використовується callback-функції.

Callback-функція – це функція зворотного виклику, яка повинна виконатись, після того, як попередня функція завершила свою роботу. В JavaScript функції є об’єктами. Через це, функції можуть приймати інші функції в якості аргументів, а також функції можуть повертати функції в якості результату. Функції, які мають таку можливість називають функціями вищого порядку. А будь-яка функція, яка передається як аргумент, називається callback-функцією.

Так як JavaScript – це подієво-орієнтована мова, то замість того, щоб чекати відповіді для подальшого виконання програми, JavaScript продовжить виконання, одночасно чекаючи інших подій. Розглянемо приклад зображений на рис. 3.1

```
function first(){
  console.log(1);
}
function second(){
  console.log(2);
}
first();
second();
```

Рисунок 3.1 – Приклад виконання синхронних команд

В результаті виконання, функція first виконається першою, а функція second після неї, тому в консолі отримаємо наступний вивід: // 1 // 2

Але що якщо функція first містить якийсь код, який не може виконатися негайно, наприклад, запит до API, де ми відправляємо запит і повинні чекати відповіді. Щоб змоделювати таку ситуацію, використовуємо функцію setTimeout, яка викликає функцію після заданого часового проміжку. Ми відстрочимо виконання функції на 500 мілісекунд, як нібито це запит до деякого API. Тепер код буде виглядати так, як зображено на рис. 3.2

```
function first(){
  setTimeout( function(){
    console.log(1);
  }, 500 );
}
function second(){
  console.log(2);
}
first();
second();
```

Рисунок 3.2 – Виконання синхронних команд з часовою затримкою

В результаті виконання, замість очікуваного виводу: // 1 // 2 – ми отримали виведення спочатку // 2, а після затримки в 500 мілісекунд - // 1, хоча й досі ми викликаємо функцію first раніше, ніж функцію second. Для отримання правильного

виводу й використовуються callback-функції. На рис. 3.3 зображено вирішення проблеми, за рахунок callback.

```
function first(callback){
  setTimeout( function(){
    console.log(1);
    callback();
  }, 500 );
}
function second(){
  console.log(2);
}
first(second);
```

Рисунок 3.3 – Використання callback-функції

Таким чином в результаті виконання даного блоку коду, після 500 мілісекунд отримаємо вивід: // 1 // 2.

Повернемося до Node та розглянемо приклад програми, зображений на рис. 3.4, для запуску простого веб-серверу. При HTTP запиті користувача в консолі буде виведено просте повідомлення: «Hello, World!».

```
var http = require('http'); // Завантажуємо модуль http

// Створюємо web-сервер і вказуємо функцію обробки запиту
var server = http.createServer(function (req, res) {
    console.log('Початок обробки запиту');
    // Передаємо код відповіді і заголовки
    res.writeHead(200, {
        'Content-Type': 'text/plain; charset=UTF-8'
    });
    res.end('Hello world!');
});

// Запускаємо web-сервер
server.listen(1991, "127.0.0.1", function () {
    console.log('Сервер запущено за адресою http://127.0.0.1:1991/');
});
```

Рисунок 3.4 – Приклад веб-серверу

У складі Node відразу міститься декілька модулів, а також встановлений власний менеджер пакунків NPM. NPM – це менеджер пакунків за замовчуванням для середовища виконання Node та для мови програмування JavaScript. В

установлених модулях відразу реалізовано типові операції для роботи з файловою системою, HTTP протоколами та обробкою даних. Для встановлення інших модулів, які є у відкритому доступі, створені іншими розробниками, можна скористатись командою:

```
npm install <packagename>
```

Для отримання інформації по доступних для встановлення модулів та отримати короткий їх опис, необхідно скористатись командою:

```
npm search
```

Цією ж командою, можна здійснювати вибірковий пошук необхідних модулів.

NPM керує пакунками, які можуть бути встановлені як залежності локального проекту, так і глобальними для системи. Зазвичай для кожного окремого проекту створюється файл package.json, який містить у собі інформацію про усі необхідні для встановлення залежності, короткий опис та назву проекту, його версію. За допомогою команди “npm install” усі необхідні пакунки відразу встановляться в папку проекту.

3.2 MongoDB

MongoDB – це документо-орієнтована система керування базами даних, яка зберігає дані у JSON-подібних документах. Вона не потребує опису схеми таблиць та зв’язку між ними. Це найбільш природний спосіб думати про дані, і він набагато виразніший та потужніший, ніж традиційна модель рядків / стовпців.

MongoDB – це справжня платформа даних із широким набором інструментів, що робить роботу з даними надзвичайно простою для всіх – від розробників до аналітиків та вчених-дослідників. В основі лежить досить гнучка мова для формування запитів до бази даних, що дозволяє з легкістю отримувати необхідні дані, об’єднувати дані з різних колекцій, створювати різного роду індекси, зберігати великі обсяги даних.

Усі данні зберігаються в бінарному BSON форматі, який є JSON-подібним документом. Цей комп’ютерний формат використовується для обміну даних. Складає собою двійкову форму представлення простих структурних даних та асоціативних масивів. BSON формат є більш ефективним, ніж JSON формат,

оскільки збереженні дані у ньому менші за розміром, а швидкість сканування – відповідно більше. Іншою особливістю є те, що великі елементи містять префікс з довжиною документа, що значно спрощує процес перебору значень.

Представимо, що у нас є колекція користувачів *users*, яка містить у собі об’єкт, зображений на рис. 3.5

Рисунок 3.5 – BSON об’єкт з інформацією про коритувача

Для того, щоб дістати цього користувача, знаючи, що він проживає за адресою «123 Main Street» необхідно виконати наступний запит.

```
db.users.find({"address.street" : "123 Main Street "});
```

Даний запит може повернути не тільки одного користувача, але й набір користувачів, які проживають за цією ж адресою. Тому для покращення фільтрування, одного параметру для пошуку може бути недостатньо.

3.3 HTML, CSS та JavaScript

HTML – це мова тегів, за допомогою якої створюються веб-сторінки – гіпертекстові документи. HTML не відноситься до мов програмування, оскільки нею неможливо здійснювати жодних математичних операцій або створити певний алгоритм.

HTML-документи зберігаються в локальній пам’яті або на веб-серверах. Браузер, отримує інформацію про те, звідки необхідно завантажити сторінку, зчитує файл і формує на його основі повноцінні розділи та веб-елементи. Структура є семантичною, тобто, представлена у вигляді орієнтованого графу. Вершини графу – це об'єкти предметної області, а ребра – відносини між об’єктами.

Основою HTML-документу є його розмітка. До неї відноситься 4 важливих компонента: елементи та їх атрибути; базові типи даних; символьні мнемоніки; декларація типу документа.

Загальна структура містить у собі три компонента: декларація типу документу, що дає інструкції для браузера про те, як саме необхідно інтерпретувати та відобразити даний документ; шапка документа, яка описує загальні технічні та додаткові відомості про документ, але не відображає його на сторінці; тіло документа є основою документа, тому браузер відображає його вміст. Розглянемо приклад структури HTML-документа, зображеного на рис. 3.6.

```
<! DOCTYPE html>
<html>
   <head>
      <title>Назва</title>
   </head>
   <body>
      <p> Hello world!</p>
   </body>
</html>
```

Рисунок 3.6 – Структура HTML-документа

Зображення, форми для введення даних користувачем, інформаційні блоки та інші об’єкти вбудовуються у візуалізовану сторінку за допомогою конструкцій HTML елементів. HTML елементи ще називають тегами. Розглянемо структуру теги.

```
<назва теги  атрибут 1="значення 1"  атрибут 2 ...>вміст теги</назва теги>
```

Назва теги вставляється у трикутні дужки, а для того, щоб браузер зрозумів, що вміст теги закінчився, необхідно окрім трикутних дужок добавити слеш (похила лінія) перед назвою теги. Окрім назви теги, у початковій тезі між трикутними дужками можна вказати ще й атрибути. За допомогою атрибутів здійснюється керування поведінкою даного елементу. Але слід враховувати, що є певні елементи, що складаються тільки з початкового тегу. До них можна віднести теги зображення, перенесення тексту на новий рядок, тощо.

Атрибути можна поділити на дві категорії: без значення та зі значенням. Для атрибутів без значень вказується тільки назва атрибуту. Атрибути зі значенням містять пару назва-значення та розділяються знаком дорівнює. Атрибути часто використовується для подальшого використання їх у мові програмування JavaScript, щоб додати до елементу нові функціональні властивості чи для додання нових візуальних властивостей за допомогою каскадних таблиць стилів CSS. Між

собою атрибути відокремлюються пробілом, а порядок їх написання ніяк не впливає на роботу.

Так як HTML унаслідується від стандартної узагальненої мови розмітки SGML, то й усі типи даних у ній також присутні. Усі елементи складаються з атрибуту та певного значення. Всі можливі варіації прописані відповідно до типів даних прописаних у декларації типу документа. Наприклад, ContentType, Charset, Script, StyleSheet.

Бувають випадки, коли необхідно вставити в HTML документ специфічні символи, які не описані в кодовій таблиці. В таких випадках використовують символьні мнемоніки – SGML посилання на символ. Мнемоніки можна поділити на два типи: цифрові та сполучення символів. Перший тип вказує на позицію символу в таблиці кодів, а другий – вказує на псевдонім символу.

Для надання сторінці певного структурованого вигляду та додавання певних візуальних ефектів використовуються каскадні таблиці стилів CSS. Це спеціальна мова, що описує візуальне представлення веб-документу. За її допомогою неможливо повноцінно написати документ або змінити його контент, але у деяких випадках можна добавити певні візуальні ефекти на сторінку, такі як спливаючі повідомлення при наведенні на певний елемент, змінити розміри певного блоку, або розділення контенту на частини для покращення його сприйняття та доступності.

Синтаксис каскадних таблиць стилів є досить простим:

```
Селектор {властивість: значення властивості}
```

Для додавання певних стилів до елементу необхідно вказати селектор елемента та додати блок визначень. Селектором елемента може виступати назва тегу, *id* або *class* елементу, назва атрибуту або його значення. Блок визначень складається з оточеного фігурними дужками списку властивостей. Властивості формуються за допомогою назви та значення властивості. Між собою властивості відділяються крапкою з комою. Приклад стилів зображено на рис. 3.7

```css
p {
  font-family: Verdana, sans-serif;
}
h2 {
  font-size: 110%;
  color: red;
  background: white;
}
.note {
  color: red;
  background: yellow;
  font-weight: bold;
}
p.warning {
  background: url(warning.png) no-repeat fixed top;
}
#paragraph1 {
  margin: 0;
}
a:hover {
  text-decoration: none;
}
#news p {
  color: red;
}
```

Рисунок 3.7 – Каскадні таблиці стилів

Щоб зробити сторінку інтерактивною чи виведення певних повідомлень для користувача, а також для здійснення операцій авторизації користувача чи завантаження контенту із серверу – необхідно використати мову програмування JavaScript. JS – об’єктно-орієнтована мова програмування, похідна від ECMAScript стандарту. Програми написані на цій мові називаються скриптами. Вони можуть вбудовуватись в HTML-документи та автоматично виконуватись, доки веб-сторінка завантажується.

Скрипти розповсюджуються та виконуються, як простий текст. Для них не потрібні певні налаштування чи компіляція для запуску. Сьогодні JavaScript може виконуватися не тільки в браузері, а й на сервері або на будь-якому іншому пристрої, який має спеціальну програму, що називається «двигуном» JavaScript. У браузера є власний двигун, який іноді називають «віртуальна машина JavaScript».

Сучасний JavaScript – це безпечна мова програмування. Вона не надає низькорівневий доступ до пам'яті або процесору, тому що спочатку була створена для браузерів, які не потребують цього.

Можливості JavaScript сильно залежать від оточення, в якому він працює. Наприклад, Node.JS підтримує функції читання / запису довільних файлів, виконання мережевих запитів і т.д. У браузері для JavaScript є все, що пов'язано з маніпулюванням веб-сторінками, взаємодією з користувачем і веб-сервером.

На рис. 3.8 зображено можливості на обмеження мови JavaScript.

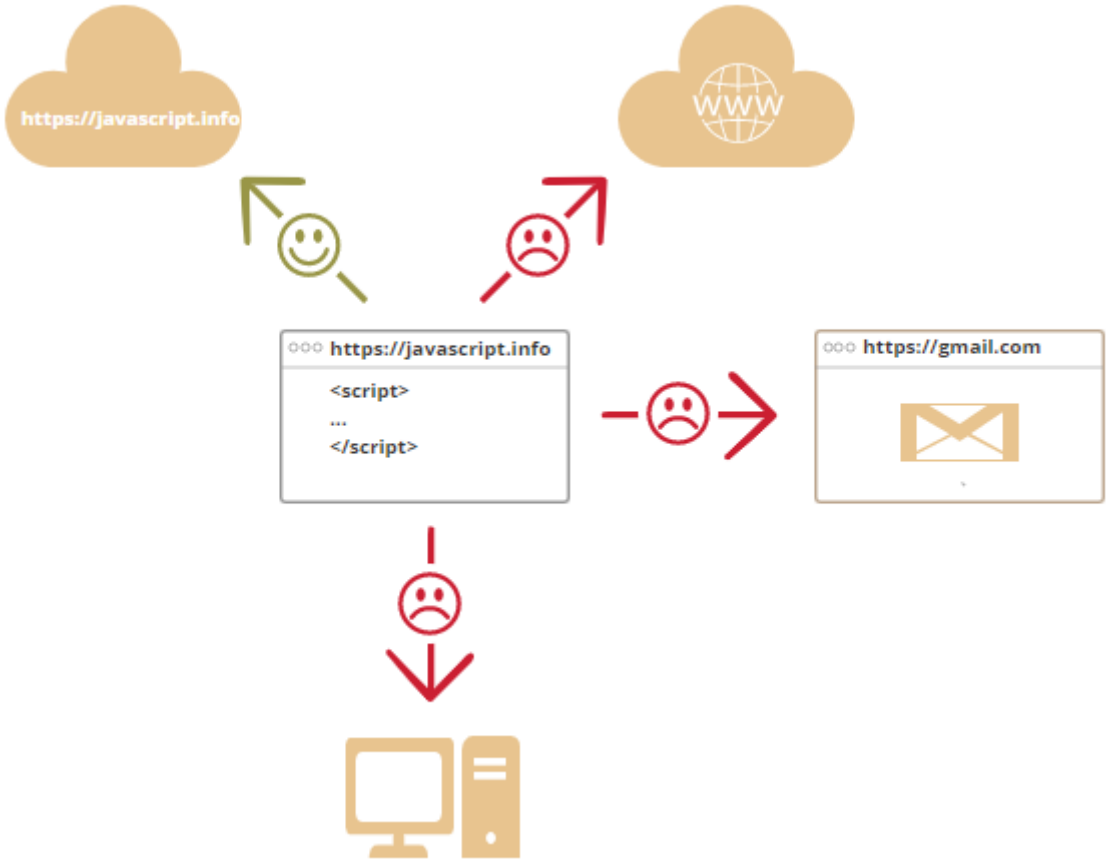


Рисунок 3.8 – Можливості та обмеження JS

Наприклад, в браузері JavaScript може:

- Додавати новий HTML-код на сторінку, змінювати існуючий вміст, модифікувати стилі;
- Реагувати на дії користувача, клацання миші, перемістити вказівник, натискання клавіш;
- Відправляти мережеві запити на віддалені сервера, завантажувати і відправляти файли (технології AJAX і COMET);
- Отримувати і встановлювати куки, задавати питання відвідувачеві, показувати повідомлення;
- Запам'ятовувати дані на стороні клієнта («local storage»).

Можливості JavaScript в браузері обмежені заради безпеки користувача. Мета полягає в запобіганні доступу недобросовісної веб-сторінки до особистої інформації або нанесення шкоди даними користувача.

Приклади таких обмежень включають в себе:

- JavaScript на веб-сторінці не може читати / записувати довільні файли на жорсткому диску, копіювати їх або запускати програми. Він не має прямого доступу до системних функцій ОС;
- Різні вікна / вкладки не знають один про одного. Іноді одне вікно, використовуючи JavaScript, відкриває інше вікно. Але навіть в цьому випадку JavaScript з однієї сторінки не має доступу до іншої, якщо вони прийшли з різних сайтів (з іншого домену, протоколу або порту);
- JavaScript може легко взаємодіяти з сервером, з якого прийшла поточна сторінка. Але його здатність отримувати дані з інших сайтів / доменів обмежена.

Подібні обмеження не діють, якщо JavaScript використовується поза браузером, наприклад – на сервері. Сучасні браузери надають плагіни / розширення, за допомогою яких можна запитувати додаткові дозволи.

3.4 WebScoket та JWT

WebSocket – це протокол, основним призначенням якого є обмін інформації в режимі реального часу між веб-сервером та браузером. Протокол забезпечує двонаправлений канал зв’язку між клієнтом та сервером.

Для встановлення з’єднання користувача та сервера необхідно створити WS об’єкт, в конструктор передається параметр WS URI та додається callback-функція при з’єднання чи розриві з’єднання, отриманні повідомлення. Приклад з’єднання зображено на рис. 3.9

```
<html>
    <head>
        <script>
            const webSocket = new WebSocket('ws://localhost/echo');

            webSocket.onopen = event => {
                alert('onopen');
                webSocket.send("Hello Web Socket!");
            };

            webSocket.onmessage = event => {
                alert('onmessage, ' + event.data);
                webSocket.close();
            };

            webSocket.onclose = event => {
                alert('onclose');
            };
        </script>
    </head>
    <body>
    </body>
</html>
```

Рисунок 3.9 – WS з’єднання

Для того, щоб ідентифікувати користувача необхідно скористатись JSON Web Token. JWT – це стандарт токена доступу, який оснований на об’єкті JSON формату. Використовується для верифікації тверджень.

JWT містить 3 частини: заголовок, вміст та підпис. В заголовку описується тип токена в змінній *typ* та методи шифрування в змінній *alg*. Приклад заголовку зображено на рис. 3.10

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Рисунок 3.10 – JWT заголовок

Вміст JWT описує певне твердження. Існують зарезервовані твердження: iss, sub, aud, exp, nbf, iat, jti. Приклад вмісту зображено на рис. 3.11

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Рисунок 3.11 – JWT вміст

Остання частина, підпис, генерується за допомогою кодування заголовку та вмісту в формат Base64, записуються через крапку в один рядок та хешується за допомогою обраного методу шифрування.

Згенерований код передається на сервер, де визначається подальші події, в залежності від того чи є даний користувач в базі даних, чи хтось намагається взломати систему.

Висновок до розділу

В даному розділі розглянуто технології, які допоможуть досягти необхідної цілі, а саме – створення серверної частини на роботизованій платформі та у сховищі, оптимізація зображень та процесу їх передачі, створення веб-додатку для представлення збережених даних користувачеві. Для розробки серверної частини обрано програмну платформу Node.JS, оскільки вона містить необхідні функціональні можливості та добре оперує з браузерами. Операції по оптимізації та передачі зображень теж будуть написані на Node, оскільки можна отримати певні готові модулі за допомогою менеджера пакунків NPM. В якості бази даних доцільно використати документо-орієнтовану систему керування базами даних MongoDB. Вона дозволяє зберігати дані у вигляді об’єктів, що значно спрощує роботу з великими об’ємами різних типів даних. Для створення клієнтської частини використовуються базові мови HTML та CSS, без додатковий препроцесорів чи бібліотек, щоб отримати максимальну швидкодію системи. Для встановлення зв’язку між клієнтом та сервером використовується WebSocket протокол, щоб надати користувачу можливість переглядати трансляцію з камери платформи. Щоб обмежити доступ до робота та сховища обрано JWT стандарт для верифікації користувача.

РОЗДІЛ 4. РОЗРОБКА СИСТЕМИ ТА ОПТИМІЗАЦІЯ ЗОБРАЖЕНЬ

4.1 Компоненти системи

Основою системи є комп’ютер Raspberry Pi 4 Model B. Raspberry Pi 4 Model B – це останній продукт у популярній лінійці комп'ютерів Raspberry Pi. Він пропонує революційне збільшення швидкості процесора, продуктивності мультимедіа, пам'яті та підключення у порівнянні з попереднім поколінням Raspberry Pi 3 Model B +, зберігаючи зворотну сумісність та подібне споживання енергії. Для кінцевого користувача Raspberry Pi 4 Model B забезпечує продуктивність робочого столу, порівнянну із системами ПК x86 початкового рівня. Комп’ютер зображено на рис. 4.1

Рисунок 4.1 – Raspberry Pi 4 Model B

Основні характеристики цього продукту включають високопродуктивний 64-розрядний чотирьох ядерний процесор, підтримку подвійного дисплея з роздільною здатністю до 4K через пару портів micro-HDMI, апаратне декодування відео до 4Kp60, до 8 Гб оперативної пам'яті, подвійний діапазон бездротової локальної мережі 2,4 / 5,0 ГГц, Bluetooth 5,0, Gigabit Ethernet, USB 3,0 та PoE (через окремий додаток PoE HAT).

Двосмугові бездротові локальні мережі та Bluetooth мають модульну сертифікацію відповідності, що дозволяє розробляти плату на кінцеві продукти зі значно зменшеним тестуванням на відповідність, покращуючи як вартість, так і час виходу на ринок.

Наступним важливим компонентом системи є камера RASPBERRY PI INFRARED. Модуль камери дуже популярний у програмах домашньої безпеки. Камера працює з усіма моделями Raspberry Pi 1, 2, 3 і 4. До неї можна отримати доступ через API MMAL та V4L, і для неї створено численні сторонні бібліотеки, включаючи бібліотеку Picamera Python.

Камера має 8-мегапіксельний датчик Sony IMX219, кут огляду 75,7 градусів, роздільну здатність 1080p, підтримує підключення інфрачервоного світлодіода або світлодіодного спалаху. Камера зображена на рис. 4.2

Рисунок 4.2 – RASPBERRY PI INFRARED

4.2 Архітектура системи

Вся система складається з веб-додатку та двох серверних частин: одна розміщується на рухомій платформі, а інша – вбудована до сховища. В свою чергу, система базується на дев’яти функціональних модулях, які забезпечують повноцінну її роботу (рис. 4.3).

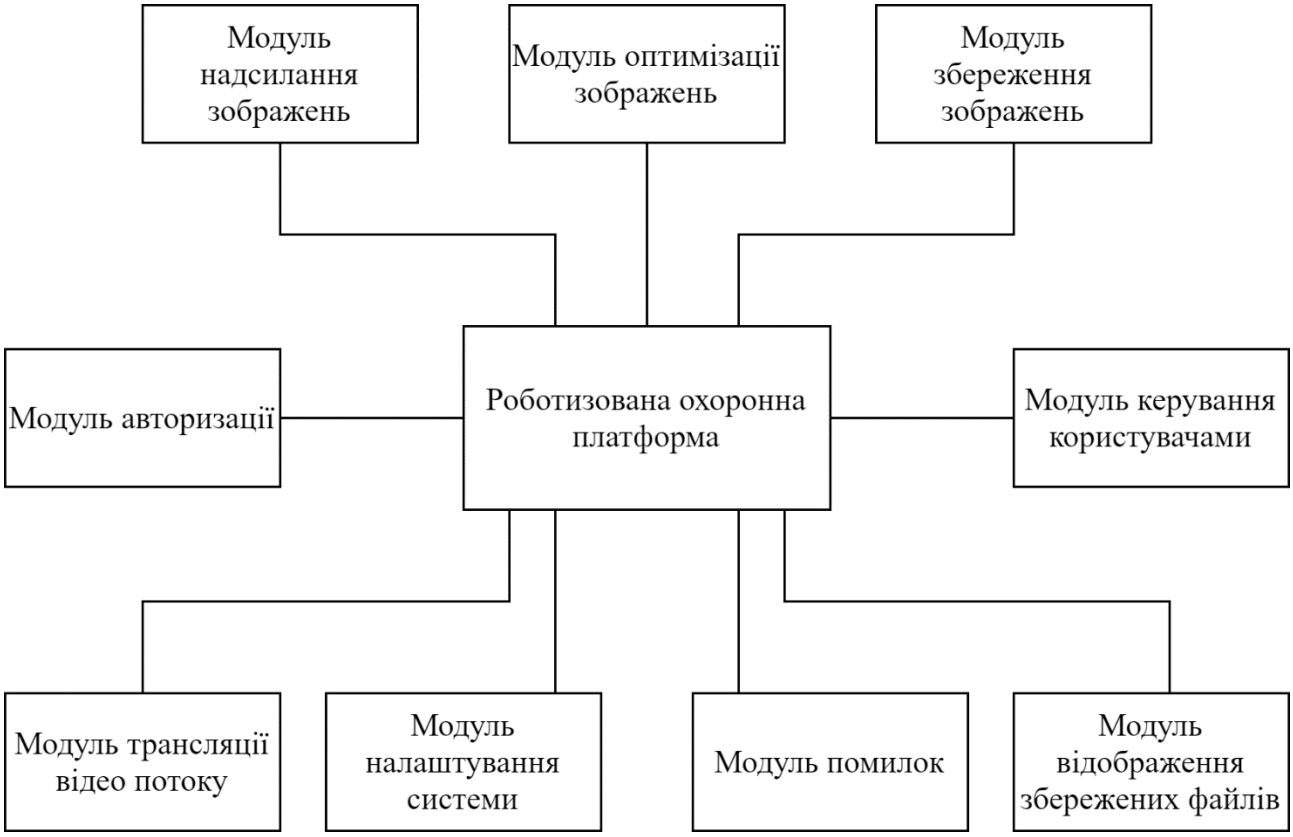


Рисунок 4.3 – Архітектура системи

Розглянемо дану структуру більш детально. Розпочати слід із модулю керування користувачів. Даний модуль надає можливість користувачеві змінити його поточні параметри, додати нового або видалити існуючого користувача. При першому заході у систему користувач повинен створити нового користувача, щоб мати змогу налаштувати параметри системи та для подальшого доступу до системи віддалено. Даний користувач є основним у системі, тобто є адміністратором системи, та має доступ до усіх налаштувань системи, може створювати нових, або переглядати та редагувати інформацію існуючих користувачів. В такому разі, усім користувачам, які будуть створюватись після цього, необхідно буде спочатку отримати підтвердження від адміністратора, яке надсилається йому на пошту. Після отримання підтвердження, профіль користувача буде створено, а на його пошту прийде повідомлення про успішне створення профілю. Новий користувач після успішної авторизації має змогу переглядати історію збережених файлів та переглядати відео трансляцію онлайн, а також може редагувати інформацію свого профілю, таку як зміна поштової адреси, оновлення чи відновлення паролю, зміна імені користувача.

Після етапу створення користувача, необхідною умовою для користування системою є процес авторизації, за що відповідає модуль авторизації. Для входу користувача у систему, йому необхідно у форму для входу ввести свої персональні дані, після чого буде згенеровано унікальний зашифрований ключ, за допомогою якого користувач буде мати змогу виконувати певні, відповідні до його ролі, функції. Якщо користувач увів не вірні дані, в такому випадку модуль помилок сповістить його про це та запропонує увести дані ще раз. Якщо ж користувач втратив дані до свого облікового запису, тоді він може натиснути на кнопку *Forgot your password? Click here*, для відновлення доступу до свого профілю через пошту, на яку було здійснено реєстрацію.

Наступний важливий модуль відповідає за процес оптимізації зображень. За його допомогою, усі збережені та готові до подальшого відправлення зображення, а також кадри відео потоку, будуть оптимізуватись шляхом конвертації файлів у формат WebP та їх стиснення за рахунок параметру якості. Адміністратор системи має змогу налаштувати параметр якості вихідного зображення та може вказати бажану кількість кадрів у секунду, які будуть зберігатись у сховищі чи транслюватись у відео потік. Щоб досягти максимальної швидкодії по передачі файлів від роботизованої платформи до серверу-сховища, даний модуль буде конвертувати фотографії в бінарний вигляд, що значно зменшить розмір переданих файлів, порівняно з форматом Base64.

Модуль надсилання зображень від платформи до сховища, в залежності від обраних адміністратором налаштувань, надсилає отримані з камери та оброблені фотографії з певним виставленим інтервалом. У свою чергу, модуль прийому та збереження зображень працює на стороні сховища. При запиті на збереження, здійснюється перевірка на наповненість сховища, при необхідності здійснюється вивільнення необхідного простору шляхом видалення старих файлів, далі дані запиту конвертуються з бінарного формату в формат WebP для зберігання. Всі збереженні файли розділяються по папкам, для полегшення їх подальшого перегляду. Основними роздільниками є папки з датою запису у форматі *DD-MM-YYYY* (день-місяць-рік), всередині кожної з яких містяться папки у форматі *HH* (година) для розділення записів по годинам. Кожна година розбивається на

проміжки по 10 хвилин кожний, відповідні папки мають формат М-М (перша та десята хвилина), всередині яких знаходяться усі зображення зроблені в певну хвилину.

За допомогою модуля трансляції відео потоку, користувач може віддалено переглядати за всім, що відбувається в приміщенні в режимі реального часу з мінімальною затримкою. Адміністратор системи має змогу налаштувати кількість кадрів, які будуть транслюватись та змінити якість зображень, які передаються. У разі, коли робот з певних причин не здійснює трансляцію відео потоку, наприклад, платформа заряджається або в даний момент відсутнє з'єднання з мережею інтернет, за допомогою модуля помилок, користувач отримає повідомлення про те, що в даний момент доступ до робота відсутній.

Останній модуль для відображення збережених файлів складається з низки доступних запитів. Кожний з них відповідає за реалізацію певного функціоналу. Перший запит *getFolders* повертає користувачу набір збережених у сховищі папок у форматі *DD-MM-YYYY* (день-місяць0рік), щоб полегшити доступність. Далі, за допомогою запиту *getDate* користувач отримує список годин, в які платформа надсилала зображення на зберігання в обраний день. Третій запит getMinutes повертає список 10-ти хвилинних інтервалів. Четвертий запит *getImages* відображає усі зображення, збережені на платформі в обраний день та час. Для відображення усіх фотографій у вигляді відео, реалізовано ще один запит *getVideo*. За його допомогою, усі завантажені фотографії конвертуються у ряд кадрів, які потім транслюються для користувача, як відео потік.

4.3 Структура бази даних

База даних системи складається з 2 колекцій *users* та *configurations*. Вони необхідні для нормального функціонування системи та для збереження необхідної інформації.

Перша колекція користувачів *users* складається з чотирьох полів: унікальна поштова скринька користувача *email*, зашифрований пароль *password*, роль *role* та ім'я користувача *name*. Існує два типи ролі *user* та *admin*. Першому створеному користувачеві автоматично буде присвоєно роль *admin*, тому даний користувач

буде мати доступ до усіх можливостей системи, а також зможе підтверджувати створення нових користувачів. Схему колекції користувачів зображено на рис. 4.4

```
const UserSchema = new mongoose.Schema({
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
  },

  password: {
    type: String,
    required: true,
  },

  name: {
    type: String
  },

  role: {
    type: String,
    default: 'user',
  }
});
```

Рисунок 4.4 – Схема колекції користувачів

Друга колекція відповідає за налаштування системи та складається з чотирьох полів: кількість кадрів у секунду для передачі відео потоку *streamFPS*, якість вихідного зображення для відео потоку *streamQuality*, кількість кадрів у секунду для збереження у сховищі *exportFPS*, якість зображення для збереження *exportQuality*. Користувач із правами адміністратора має змогу налаштувати ці параметри. Схему колекції налаштувань системи зображено на рис. 4.5

```
const ConfigurationSchema = new mongoose.Schema({
  streamFPS: {
    type: Number,
    default: 24
  },

  streamQaulity: {
    type: Number,
    default: 1.0
  },

  exportFPS: {
    type: Number,
    default: 24
  },

  exportQaulity: {
    type: Number,
    default: 1.0
  }
});
```

Рисунок 4.5 – Схема налаштувань системи

4.4 Розробка API

Розпочати розробку системи необхідно з розробки кінцевих точок доступу API. Дана система є закритою, тобто доступ до системи мають тільки зареєстровані користувачі. Щоб отримати доступ, користувач повинен здійснити вхід у систему та отримати згенерований веб-токен, що необхідний для усіх запитів системи. При переміщенні користувача між сторінками здійснюється контроль чи пройшов користувач автентифікацію та чи має він відповідні права користувача для відвідування певної сторінки. Для цього використовується сервіс *middlewares*, який на кожному запиті здійснює відповідну перевірку. При успішному проходженні даного етапу для користувача буде відображена необхідна інформація, а якщо з певних причин процес авторизації буде провалено, то для користувача буде виведено відповідне повідомлення про помилку.

У системі використовуються HTTP та WebSocket запити. WebSocket запити дуже добре справляються з передачею відео трансляції від камери до користувача в реальному часі. Насправді, відео потік складається з набору великої кількості зображень, які передаються від сервера до браузера. Звичайний HTTP запит дає змогу надсилати та отримувати таку кількість інформації, але для її коректного відображення необхідно створювати інтервали на стороні клієнта, що значно зменшує швидкодію системи. WebSocket сервер разом із перевіркою користувача на авторизацію зображено на рис. 4.6 та 4.7

```
const WEBSOCKET_PORT = process.argv[4] || 8084,
const socketServer = new (require('ws').Server)({port: WEBSOCKET_PORT});

const WebSocketServer = require('ws').Server;
const socketServer = new WebSocketServer({
    verifyClient: function (info, cb) {
        var token = info.req.headers.token
        if (!token)
            cb(false, 401, 'Unauthorized')
        else {
            jwt.verify(token, 'secret-key', function (err, decoded) {
                if (err) {
                    cb(false, 401, 'Unauthorized')
                } else {
                    info.req.user = decoded //[1]
                    cb(true)
                }
            })

        }
    }
});
```

Рисунок 4.6 – WebSocket сервер

```
socketServer.on('connection', function(socket) {
    // Send magic bytes and video size to the newly connected socket
    // struct { char magic[4]; unsigned short width, height;}
    var streamHeader = new Buffer(8);
    streamHeader.write(STREAM_MAGIC_BYTES);
    streamHeader.writeUInt16BE(width, 4);
    streamHeader.writeUInt16BE(height, 6);
    socket.send(streamHeader, {binary:true});

    console.log( 'New WebSocket Connection ('+socketServer.clients.length+' total)' );

    socket.on('close', function(code, message){
        console.log( 'Disconnected WebSocket ('+socketServer.clients.length+' total)' );
    });
});
```

Рисунок 4.7 – WebSocket сервер

HTTP запити розділені на дві частини. Перша частина відповідає за відображення візуальної частини. Тобто, при переході користувача на певну сторінку, здійснюється виклик певного запиту, який у свою чергу поверне HTML-документ з підключеними JavaScript скриптами та стилями. Приклад реалізації зображено на рис. 4.8. Дані запити покривають усі сторінки, на які авторизований користувач має змогу зайти.

```
app.get('/login', function(req, res){
    res.sendFile('./login.html', {root: __dirname});
});

app.get('/cameraStream', function(req, res){
    res.sendFile('./cameraStream.html', {root: __dirname});
});

app.get('/storageHistory', function(req, res){
    res.sendFile('./storageHistory.html', {root: __dirname});
});

app.get('/systemConfiguration', function(req, res){
    res.sendFile('./systemConfiguration.html', {root: __dirname});
});

app.get('/users', function(req, res){
    res.sendFile('./users.html', {root: __dirname});
});

app.get('/usersSettings', function(req, res){
    res.sendFile('./usersSettings.html', {root: __dirname});
});
```

Рисунок 4.8 – HTTP-запити для відображення сторінки

Друга частина запитів відповідає за взаємодію користувача із системою. Дані запити необхідні для створення, редагування або видалення користувача. Також ці методи застосовуються для налаштування системи, передачу та збереження зображень у сховищі, отримання файлів та їх наступне відображення у веб-

сторінці. Типова структура HTTP запитів для роботи з користувачами зображена на рис. 4.9

```
app.post('/user/login', async (req, res) => {
  const email = req.body.user.email;
  const password = req.body.user.password;
  try {
    const authServiceInstance = new AuthService();
    const { user, token } = await authServiceInstance.Login(email, password);
    return res.status(200).json({ user, token }).end();
  } catch(e) {
    return res.json(e).status(500).end();
  }
});

app.post('/user/login-as', isAuth, attachCurrentUser, checkRole('admin'), async (req, res) => {
  try {
    const email = req.body.user.email;
    const authServiceInstance = new AuthService();
    const { user, token } = await authServiceInstance.LoginAs(email);
    return res.status(200).json({ user, token }).end();
  } catch(e) {
    console.log('Error in login as user: ', e);
    return res.json(e).status(500).end();
  }
});

app.post('/user/signup', async (req, res) => {
  try {
    const { name, email, password } = req.body.user;
    const authServiceInstance = new AuthService();
    const { user, token } = await authServiceInstance.SignUp(email, password, name);
    return res.json({ user, token }).status(200).end();
  } catch (e) {
    return res.json(e).status(500).end();
  }
});
```

Рисунок 4.9 – Структура HTTP запитів для роботи з користувачем

4.5 Розробка серверної частини системи

4.5.1 Модуль авторизації та керування користувачами

Оскільки дана система є закритого типу, тобто при всіх діях користувача здійснюється перевірка веб-токена, то необхідно розглянути систему авторизації більш детально.

При створенні нового облікового запису з форми реєстрації до колекції передається інформація про користувача, його поштова скринька, пароль та ім’я. Цей пароль необхідно зашифрувати та зберегти в базі даних. Найкращим методом шифрування є метод Argon2. Разом з паролем в базу даних зберігається електронна адреса та його ім’я. Параметр ролі за замовчуванням виставляється *user*, але в процесі підтвердження адміністратором системи роль може бути зміненою на *admin*. На рис. 4.10 зображено процес створення нового користувача.

```
import * as argon2 from 'argon2';

class AuthService {
  public async SignUp(email, password, name): Promise<any> {
    const passwordHashed = await argon2.hash(password);

    const userRecord = await UserModel.create({
      password: passwordHashed,
      email,
      name,
    });

    const token = this.generateJWT(userRecord);

    return {
      user: {
        email: userRecord.email,
        name: userRecord.name,
      },
      token
    }
  }
}
```

Рисунок 4.10 – Створення нового користувача

Схема дій, які виконуються в тому випадку, коли користувач намагається увійти в систему, зображено на рис. 4.11

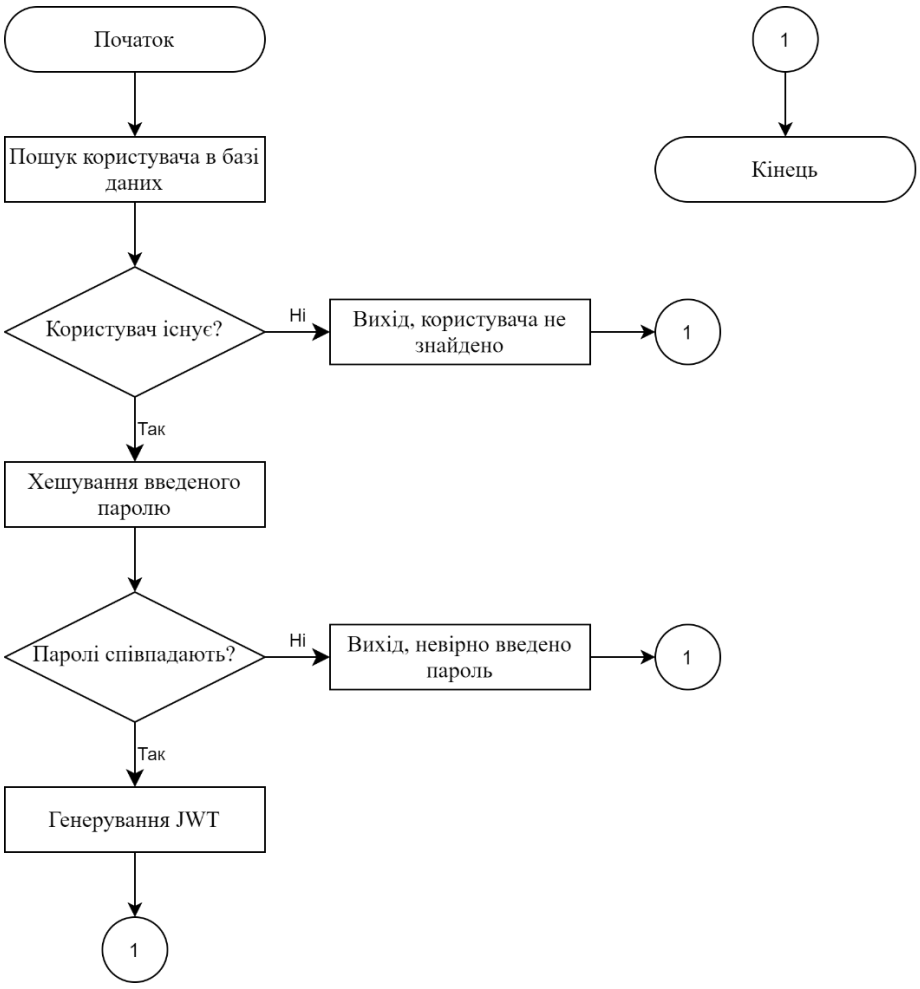


Рисунок 4.11 – Алгоритм авторизації користувача

Ось що відбувається при вході користувача в систему:

- Клієнт відправляє серверу комбінацію, що складається з публічного ідентифікатора і приватного ключа користувача. В нашому випадку це – адреса електронної пошти та пароль;
- Сервер шукає користувача в базі даних за адресою електронної пошти;
- Якщо користувач існує в базі даних - сервер шифрує відправлений йому пароль і порівнює те, що вийшло, з хешем пароля, збереженим в базі даних;
- Якщо перевірка виявляється успішною - сервер генерує так званий токен або маркер автентифікації – JSON Web Token (JWT).

JWT – це тимчасовий ключ. Клієнт повинен відправляти цей ключ серверу з кожним запитом до кінцевої точки. На рис. 4.12 зображено процес входу та отримання JWT токену.

```
import * as argon2 from 'argon2';

class AuthService {
  public async Login(email, password): Promise<any> {
    const userRecord = await UserModel.findOne({ email });
    if (!userRecord) {
      throw new Error('User not found')
    } else {
      const correctPassword = await argon2.verify(userRecord.password, password);
      if (!correctPassword) {
        throw new Error('Incorrect password')
      }
    }


    return {
      user: {
        email: userRecord.email,
        name: userRecord.name,
      },
      token: this.generateJWT(userRecord)
    }
  }
}
```

Рисунок 4.12 – Авторизація користувача

Верифікація пароля проводиться з використанням бібліотеки argon2. Це робиться для запобігання так званих «атак по часу». При виконанні такої атаки зловмисник намагається зламати пароль методом грубої сили, ґрунтуючись на аналізі того, скільки часу потрібно для сервера на формування відповіді.

Для генерації токена створено функцію *generateToken*, яка потрібна для завершення роботи над сервісом аутентифікації користувачів *middlewares*. JWT-токен сформовано за допомогою бібліотеки jsonwebtoken для Node.JS. Функцію генерації представлено на рис. 4.13

```
import * as jwt from 'jsonwebtoken'
class AuthService {
  private generateToken(user) {

    const data =  {
      _id: user._id,
      name: user.name,
      email: user.email
    };
    const signature = 'wgUsh20aG';
    const expiration = '6h';

    return jwt.sign({ data, }, signature, { expiresIn: expiration });
}
```

Рисунок 4.13 – Функція генерації JWT-токена

Тепер клієнтському коду потрібно відправляти JWT в кожному запиті до захищеної кінцевої точки. Рекомендується включати JWT в заголовки запитів. Зазвичай їх включають в заголовок Authorization. Тепер на сервері потрібно створити код, який представляє собою проміжне програмне забезпечення для маршрутів express. Приклад зображено на рис. 4.14

```
import * as jwt from 'express-jwt';

const getTokenFromHeader = (req) => {
  if (req.headers.authorization && req.headers.authorization.split(' ')[0] === 'Bearer') {
    return req.headers.authorization.split(' ')[1];
  }
}

export default jwt({
  secret: 'wgUsh20aG',
  userProperty: 'token',
  getToken: getTokenFromHeader
});
```

Рисунок 4.14 – Проміжне програмне забезпечення для авторизації

Корисно мати можливість отримувати повні відомості про обліковий запис користувача з бази даних і приєднувати їх до запиту. У нашому випадку ця можливість реалізується засобами проміжного програмного забезпечення. Його спрощений код представлено на рис. 4.15

```
export default (req, res, next) => {
  const decodedTokenData = req.tokenData;
  const userRecord = await UserModel.findOne({ _id: decodedTokenData._id });

  if(!userRecord) {
    return res.status(401).end('User not found')
  } else {
    req.currentUser = userRecord;
    return next();
  }
}
```

Рисунок 4.15 – Отримання відомостей про користувача

Після реалізації цього механізму маршрути зможуть отримувати відомості про користувача, який виконує запит. Тепер усі маршрути, окрім сторінки авторизації є захищеними. Для доступу до кінцевих точок користувач повинен мати валідний JWT. Маршрути, крім того, можуть використовувати відомості про користувача для пошуку в базі даних необхідних відомостей.

4.5.2 Модуль оптимізації та передачі зображень

Одним з основних модулів системи є модуль для стиснення розміру зображень та конвертація файлів у бінарний вигляд для надсилання до сховища чи трансляція для користувача. Перш за все, при включенні робота, сервер здійснює підключення до камери на платформі, для запису, конвертації та передачі відео потоку у необхідні файли. Для цього використано NPM бібліотеку *FFmpeg*, яка за допомогою однієї команди здійснює усе необхідне.

Важливо відзначити, що у момент запуску системи, здійснюється зчитування налаштувань з бази даних, а саме максимальна кількість кадрів у секунду, з якою необхідно зчитувати зображення з камери, після чого ці параметри застосовуються. При записі з камери здійснюється розподіл на дві частини. Перша частина записує файли з вказаною частотою *exportFPS* у пам'ять робота, для подальшої роботи з ними. Усі файли передаються у форматі image2, який формує та зберігає зображення у форматі jpg.

Команда запуску для запису фотографій у пам'ять робота:

```
ffmpeg -f dshow -rtbufsize 100M -i video=Camera abc.mp4 -r ${exportFPS} -f image2 ./savedImages/input.mp4_fr%7d.jpg
```

Друга частина веде запис з камери та перетворює кадри у відео потік, який потім транслюється на створений WebSocket сервер. Вихідні файли передаються у

форматі mjpeg, а за допомогою передачі параметру якості *streamQuality,* усі кадри автоматично будуть стискуватись за алгоритмом JPEG, що дозволить не втрачати час при конвертації зображення у інший формат. У команду також передається частота здійснення кадрів за допомогою параметра *StreamFPS*.

Команда запуску трансляції відео потоку на WebSocket сервер:

```
ffmpeg -rtbufsize 100M -i /dev/video0 abc.mp4 –r ${streamFPS} –q:v ${streamQuality} -f mjpeg http://localhost:8082/12345/1920/1080
```

Використання цих параметрів та відповідні налаштування з боку користувача під свої потреби та пропускні можливості інтернету, надає можливість транслювати відео потік з мінімальними затримками для декількох користувачів одночасно, а також досягти балансу між якістю, кількістю кадрів та затримкою в кінцевому результаті.

Оскільки MJPEG формат дозволяє автоматично стискувати розмір усіх кадрів, то необхідним в даному процесі залишається процес оптимізації передачі усіх кадрів через WebSocket сервер до усіх користувачів. Для цього необхідно всі кадри записувати у буфер та перетворювати в двійкову систему, що значно зменшує розмір вихідного рядка для подальшого передавання до клієнта. Розглянемо розміри рядків при перетворенні у бінарний вигляд та у форматі Base64.

При використанні формату Base64, розмір вихідного рядка в середньому складає близько 42500 байтів для кожного окремого кадру (табл. 4.1).

Таблиця 4.1 – Розміри рядків кожного кадру у форматі Base64

Кадр	Довжина рядка
1	42455 (41.5 kB)
2	42735 (41.7 kB)
3	42751 (41.7 kB)
4	43075 (42.0 kB)
5	42451 (41.5 kB)

Розглянемо процес передачі за допомогою конвертації даних у буфер у вигляді бінарних даних. Для цього створено сервер, який при отриманні кожного кадру здійснює його конвертацію у бінарний вигляд та надсилає їх на створений WebSocket сервер для передачі відео потоку до усіх підключених клієнтів. Код серверу зображено на рис. 4.16

```
const WEBSOCKET_PORT = process.argv[4] || 8084,
const socketServer = new (require('ws').Server)({port: WEBSOCKET_PORT});

const WebSocketServer = require('ws').Server;
const socketServer = new WebSocketServer({
    verifyClient: function (info, cb) {
        var token = info.req.headers.token
        if (!token)
            cb(false, 401, 'Unauthorized')
        else {
            jwt.verify(token, 'secret-key', function (err, decoded) {
                if (err) {
                    cb(false, 401, 'Unauthorized')
                } else {
                    info.req.user = decoded //[1]
                    cb(true)
                }
            })

        }
    }
});
socketServer.on('connection', function(socket) {
    // Send magic bytes and video size to the newly connected socket
    // struct { char magic[4]; unsigned short width, height;}
    var streamHeader = new Buffer(8);
    streamHeader.write(STREAM_MAGIC_BYTES);
    streamHeader.writeUInt16BE(width, 4);
    streamHeader.writeUInt16BE(height, 6);
    socket.send(streamHeader, {binary:true});

    console.log( 'New WebSocket Connection ('+socketServer.clients.length+' total)' );

    socket.on('close', function(code, message){
        console.log( 'Disconnected WebSocket ('+socketServer.clients.length+' total)' );
    });
});
```

Рисунок 4.16 – Код серверу для передачі та конвертації

В результаті, розміри рядків усіх кадрів мають значно менші розміри, майже в третину, ніж при використанні конвертації зображень у формат Base64 (табл. 4.2).

Таблиця 4.2 - Розміри рядків кожного кадру в бінарному форматі

Кадр	Розмір рядка
1	27.5 kB
2	27.7 kB
3	28.0 kB
4	27.7 kB
5	27.8 kB

Як вже було відзначено раніше, частина кадрів зберігається у пам'ять робота, для подальшої їх оптимізації, передачі до сховища та збереження їх там. Розглянемо цей процес більш детально.

При зчитуванні кадрів з камери, в залежності від налаштувань системи, частина з них зберігається у форматі jpg, оскільки даний формат зберігає зображення у хорошій якості. Проблемою є те, що розміри таких файлів є досить

великими, тому необхідно здійснити їх стиснення. Для цього використано NPM бібліотеки *imagemin* та *imagemin-webp*, за допомогою яких можна здійснити конвертацію зображень з формату jpg у формат webp без втрати якості, оскільки зображення в даному форматі при такій самій якості значно менші за розміром, а за допомогою налаштувань можна вказати кінцеву якість фотографії, щоб ще більше зменшити розміри кінцевого файлу. Приклад реалізації зображено на рис. 4.16

```
const imagemin = require('imagemin');
const imageminWebp = require('imagemin-webp');

const exportQuality = mongo.get('exportQuality');

imagemin(['images/*'], {
  destination: 'compressed_images',
  plugins: [imageminWebp({quality: exportQuality})]
}).then(() => {
  sendFiles();
});
```

Рисунок 4.17 – Конвертація та стиснення зображення

Порівняємо розміри зображень без стиснення у форматі JPG до перетворення та у WebP після процесу конвертації, а також порівняємо розміри файлів WebP до та після зменшення якості. Усі результати наведені нижче (табл. 4.3). При конвертації файлів, розмір кінцевих файлів трохи зменшився у порівнянні з оригінальними зображеннями. При використанні алгоритму стиснення WebP та якості 80%, помітити різницю між оригінальним та стиснутим зображенням практично неможливо, хоча розмір зображення зменшується майже вдвічі. При якості 50%, різниця помітна трохи більше, але не є критичною, а розміри файлів зменшуються майже втричі.

Таблиця 4.3 – Порівняння розмірів зображень

JPG	WebP (100%)	WebP (80%)	WebP (50%)
108 kB	90 kB	44 kB	29 kB
108 kB	90 kB	44 kB	29 kB
106 kB	89 kB	43 kB	29 kB
106 kB	89 kB	43 kB	29 kB
107 kB	93 kB	45 kB	30 kB

Наступним кроком є процес оптимізації передачі стиснених та перетворених зображень у форматі WebP від робота до сховища. Тут, як і в процесі передачі відео потоку, усі зображення необхідно конвертувати в бінарний вигляд, оскільки дані

запити значно менші за розмірами, у порівнянні з іншими. Метод конвертації даних у бінарний формат досить схожий до методу, використаного при передачі відео потоку, але метод передачі використовується інакший. Кожної секунди зчитуються усі оптимізовані за цей час зображення, конвертуються в бінарний формат, після чого кожний файл асинхронно відправляється на сервер-сховище. Асинхронне надсилання файлів не завантажує сервер, а процес передачі стає в рази швидшим, оскільки можна надсилати наступний файл, не очікуючи, поки попередній файл буде надіслано (рис. 4.18).

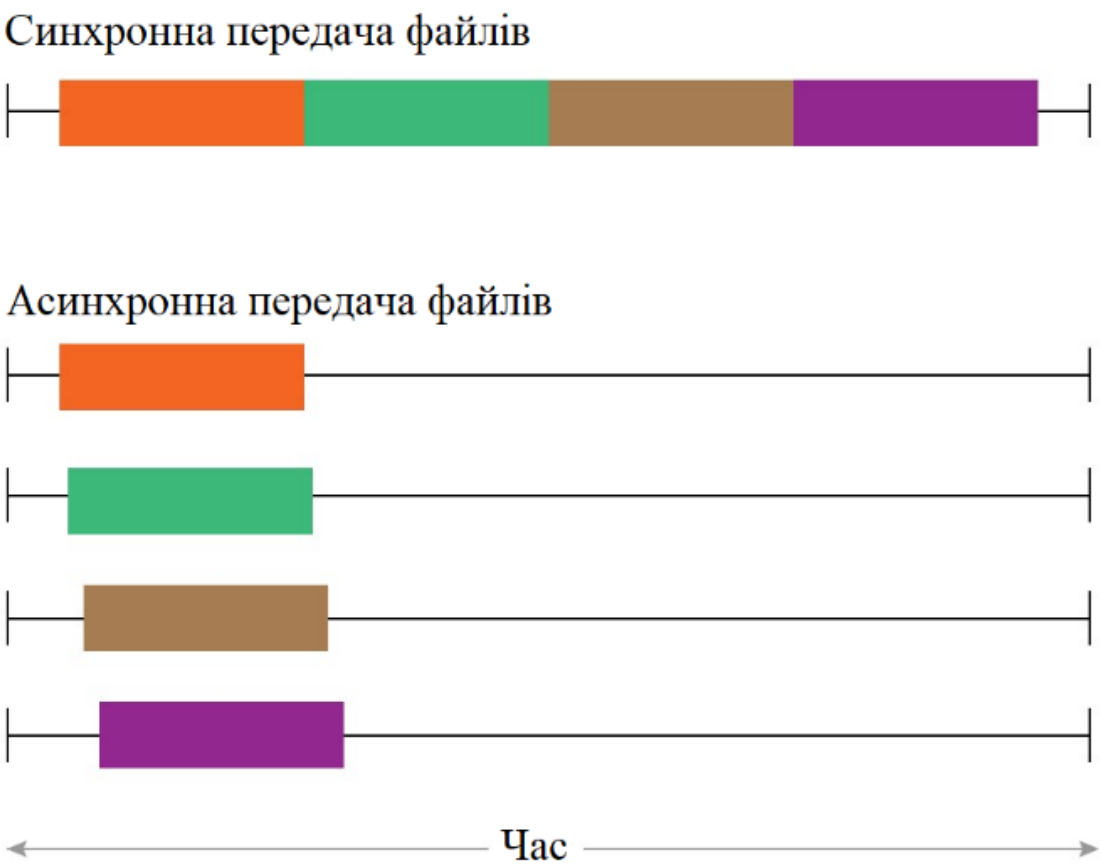


Рисунок 4.18 – Порівняння синхронної та асинхронної передачі файлів

Для отримання та збереження фотографій у сховищі, створено кінцеву точку, при запиті на яку, виконується конвертація та запис файлу. В залежності від часу та дати збереження зображення, файл розподіляється у відповідні папки. Якщо папки з необхідною датою або часом не існує, створюється нова. Алгоритм отримання та зберігання файлу зображено на рис. 4.19

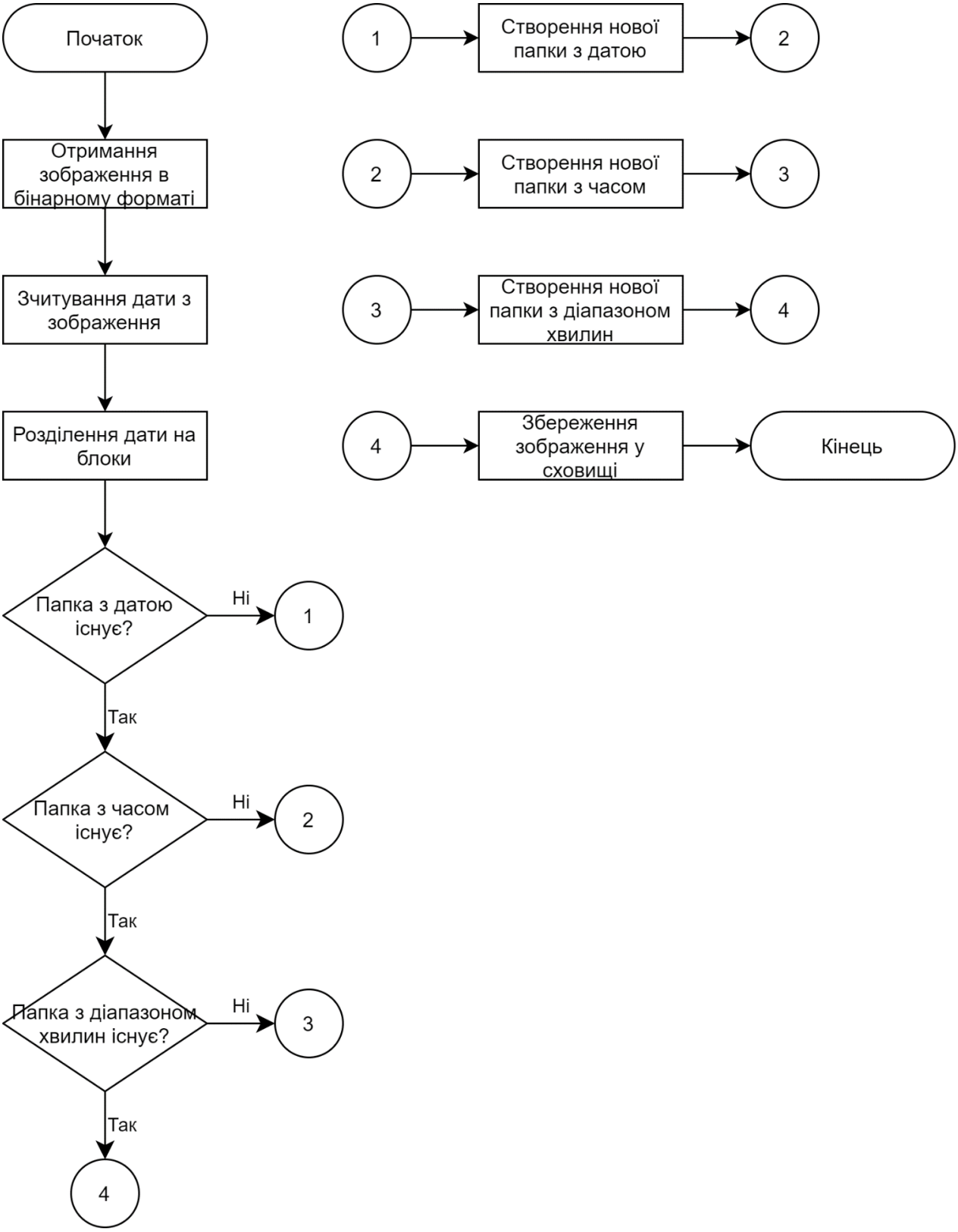


Рисунок 4.19 – Алгоритм отримання та зберігання зображення

4.6 Розробка клієнтської частини

Розробку клієнтської частини необхідно розпочати з розподілу всього додатку на різні компоненти, кожний з яких відповідає за певну частину функціоналу. Це дозволяє краще та легше підтримувати продукт в майбутньому,

дозволяє з легкістю добавляти нові функціональні можливості. Формально, можна розділити додаток на дві складові: візуальна частина та функціональні можливості.

Перша, візуальна складова, відповідає за стилістику та наповненість сторінки Тобто, відображення всього вмісту сторінки таким чином, щоб користувачу було зручно користуватись веб-системою. Для цього необхідно не навантажувати сторінку непотрібною інформацією, а всі окремі компоненти розділити на відповідні їм блоки.

У свою чергу, функціональні можливості відповідають за взаємодію користувача із системою. Наприклад, при натисненні певної кнопки буде надіслано запит до серверу, а при отриманні інформації – здійснено її відображення.

Оскільки для доступу до системи, користувачу потрібно авторизуватись, то розглянемо сторінку входу. На даній сторінці знаходиться форма для входу користувача у систему, кнопка для відновлення паролю та логотип компанії. При невірному введені поштової скриньки або паролю, буде виведено відповідне повідомлення. Для цього здійснюється перевірка введених даних з даними, які зберігаються в базі даних. Сторінка не навантажена лишньою інформацією та має приємний вигляд (рис. 4.20).

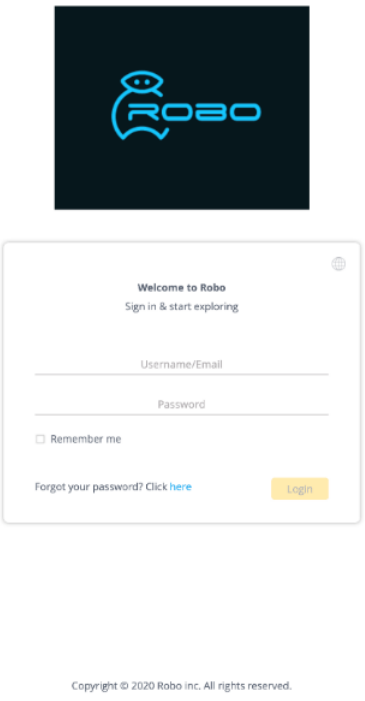


Рисунок 4.20 – Сторінка авторизації користувача

Після успішної авторизації, користувача перенесе на головну сторінку системи. У верхній частині сторінки розташовується три розділи: *Storage History*, *Robot Stream, Management Page*. В залежності від ролі користувача, в останньому розділі певні компоненти будуть відсутніми. Розглянемо кожний розділ більш детально.

На сторінці *Robot Stream* користувач має можливість слідкувати за тим, що відбувається в приміщенні в режимі реального часу, якщо робот в даний момент включений та в нього є підключення до мережі інтернет. Якщо ж від робота з певних причин не надходить сигнал, в такому випадку на сторінці буде виведено модальне вікно з вказаною причиною помилки. Для цього здійснюється підключення до серверу через WebSocket та очікується протягом кількох секунд на відповідь. Якщо в даний момент робот транслює відео потік, в такому випадку буде виведено зображення користувачеві. На рис. 4.21 представлено сторінку з трансляцією відео потоку.

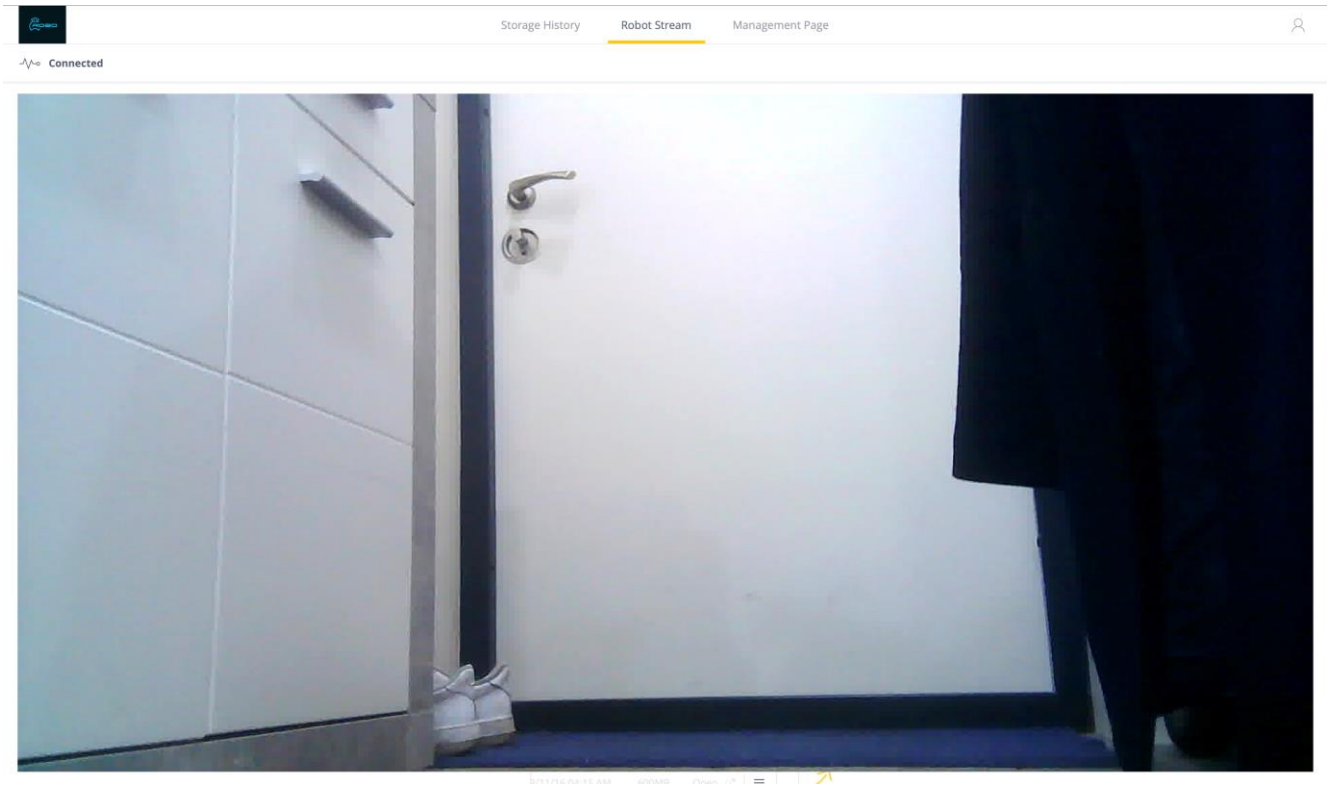


Рисунок 4.21 – Сторінка трансляції відео потоку

Сторінка *Storage History* надає можливість переглядати історію всіх кадрів, збережених у сховищі. У випадку, коли до сховища немає підключення або не знайдено збережених файлів, то для користувача буде виведено відповідне повідомлення. При знаходженні збережених файлів, у лівій частині формується

список з відображенням елементів у форматі *DD-MM-YYYY*. Кожний елемент списку відповідає за відповідну папку в сховищі, яка містить необхідні файли. При натисненні на певний елемент списку, здійснюється відправка запиту для отримання всіх папок, які відповідають вибраній даті. При надходженні відповіді в даному елементі списку формується новий – зі збереженим часом у форматі *HH*. Натиснувши на необхідний час, буде сформовано список десяти хвилинних відрізків, натиснувши на які, у головній частині сторінки сформується список усіх збережених за цей час кадрів (рис. 4.22).

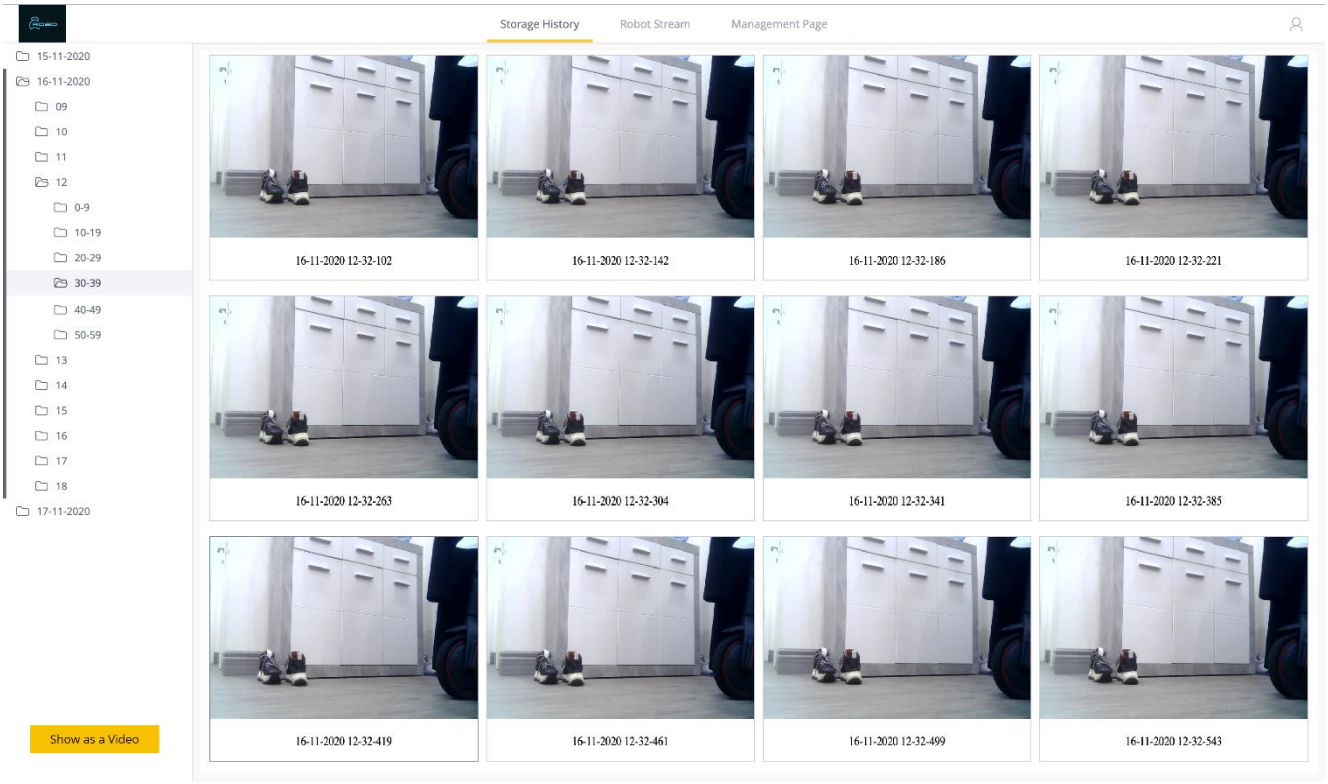


Рисунок 4.22 – Сторінка відображення вмісту сховища

Оскільки перегляд усіх кадрів окремо дещо ускладнює сприйняття, створено спеціальну кнопку *Show as a Video*, яка відкриває модальне вікно та з усіх завантажених за певний проміжок часу кадрів створює неперервне відео. Насправді, здійснюється зчитування інтервалу між кожним окремим кадром, а потім – викликається функція *showVideo* з заданим інтервалом, яка підставляє наступний у списку кадр. Для користувача весь цей процес буде непомітним, і буде здаватися, що транслюється відео. Щоб мати можливість перемотувати відео вперед або назад, створено відповідні кнопки, при натисненні на які, змінна з індексом наступного кадру змінюється на необхідну кількість кадрів.

Формула розрахунку інтервалу між окремими кадрами:

$$frameInteval = \sum_{k=0}^{n-1} \frac{(frame_{k+1} - frame_k)}{n}, \quad (4.1)$$

де *frame* – дата кадру в мілісекундах; *n* – загальна кількість кадрів.

Формула для розрахунку зміщення індексу кадру має наступний вигляд:

$$frameIndexOffset = timeOffset \times \frac{1}{frameInterval}, \quad (4.2)$$

де *frameIndexOffset* – кількість кадрів для зміщення; *timeOffset* – час у секундах, на який необхідно перемотати відео (за замовчуванням 5 секунд); *frameInterval* – інтервал між окремими кадрами.

Останньою сторінкою є *Management Page*. В залежності від ролі користувача на сторінці буде зображено один або два розділи на лівій панелі: *User Management* та *System Configuration*. Так, у звичайного користувача буде тільки розділ по редагуванні свого профілю. Адміністратор системи ж бачить усіх користувачів та може редагувати або додавати нових користувачів, а також може налаштовувати систему.

В розділі *User Management* для адміністратора відображається інформація про всіх користувачів системи. Для додавання нового користувача необхідно натиснути на кнопку *Add User* в результаті чого, відкриється модальне вікно з формою введення для додавання нового користувача у систему. Якщо необхідно редагувати певного користувача, справа від кожного користувача знаходиться значок редагування, при натисненні на який відкриється модальне вікно з формою редагування. Для видалення певного користувача необхідно натиснути на значок видалення та підтвердити свій вибір. Сторінка *User Management* представлена на рис. 4.23

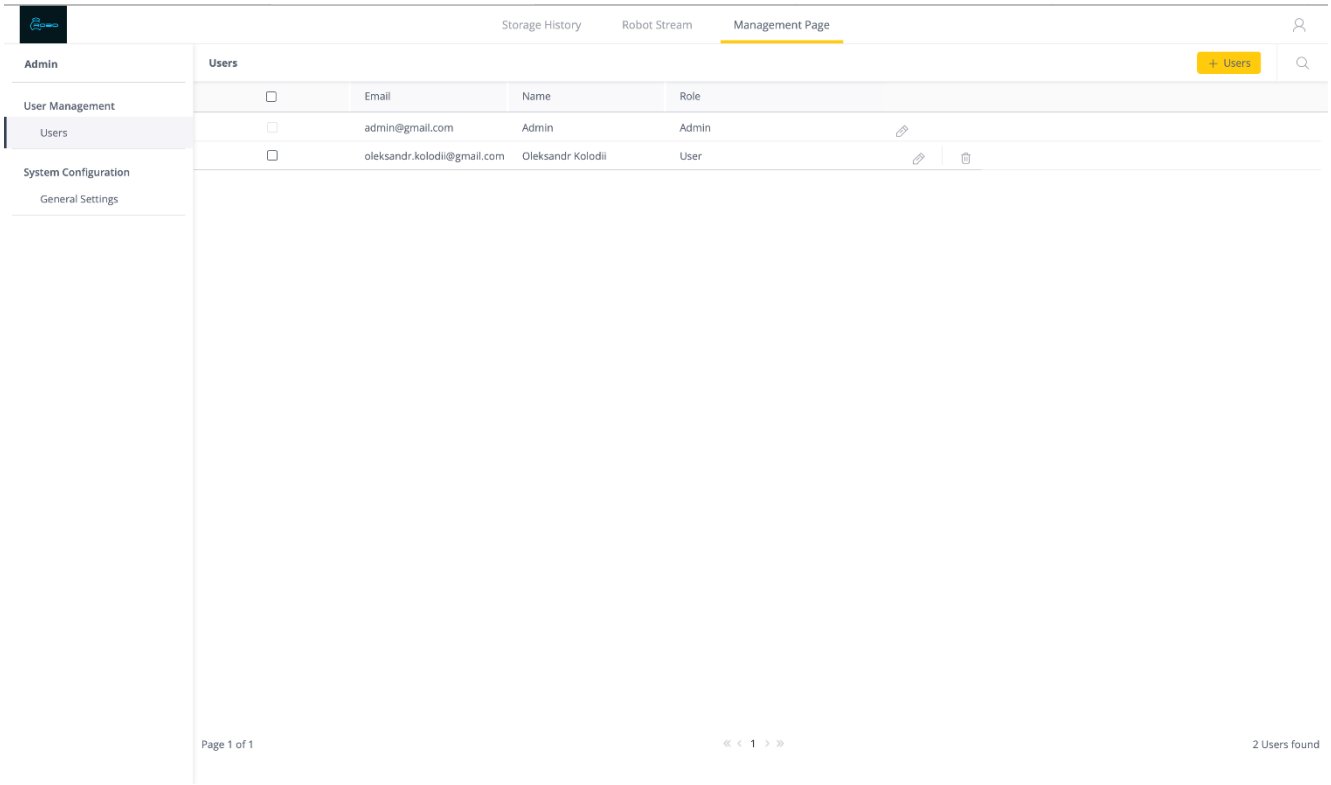


Рисунок 4.23 – Сторінка керування користувачами

В розділі *System Configuration* знаходяться налаштування системи. Тут адміністратор має можливість змінити якість зображення та кількість кадрів у секунду для трансляції відео потоку чи збереження до сховища.

Важливо відмітити, що важливою задачею є створення веб-додатку для різних пристроїв. Щоб вирішити цю проблему, було створено адаптивний веб-дизайн, тобто такий дизайн, що забезпечує оптимальне відображення та взаємодію сайту з користувачем незалежно від роздільної здатності та формату пристрою, з якого здійснюється перегляд сторінки.

Висновок до розділу

В даному розділі спроектовано архітектуру системи та описано основні компоненти. Визначено всі необхідні модулі для нормального функціонування системи в цілому та встановлено логіку взаємодії їх між собою.

Розроблено структуру бази даних для зберігання налаштувань системи та інформації про всіх користувачів. Розроблено необхідні кінцеві точки API, для взаємодії з базою даних, а також, для взаємодії веб-додатку з сервером. Створено серверну частину для робота та сховища.

Оптимізовано процес передачі всіх зображень від камери робота до кінцевого користувача та сховища. Також, здійснено стиснення розміру зображень та кадрів відео потоку, для покращення швидкодії системи та збільшення загальної збереженої інформації у сховищі.

Розроблено клієнтську частину, для відображення всіх збережених фотографій у сховищі та для трансляції відео потоку з камери робота в режимі реального часу.

РОЗДІЛ 5. МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЕКТУ

Проведемо маркетинговий аналіз стартап-проекту, щоб визначити його потенціальні можливості в процесі провадження на ринку.

5.1 Опис ідеї проекту

Розробка системи з оптимізованою передачею стиснених зображень та кадрів відео потоку від роботизованої рухомої платформи до сховища та користувача є головною ідеєю даного стартап-проекту. Одним із головних факторів системи є забезпечення хорошої швидкості роботи та зручності в користуванні користувачами. При дотриманні цих факторів, система повинна витримувати високі навантаження при одночасному використанні кількома користувачами.

Далі потрібно більш детально розглянути основні напрямки застосування та вигоду для користувачів (табл. 5.1).

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигода для користувача
Розробити систему захисту приміщень з оптимізованою передачею стиснених зображень від робота до клієнта	Застосування на підприємствах із сучасним обладнанням і високим рівнем автоматизації виробництва	Користувачі отримують зручний програмний продукт, що допомагає стежити за приміщення віддалено в режимі реального часу та зберігати необхідні дані на сервері для подальшого їх використання
	Застосування в приватних будинка чи квартирах	

Далі необхідно визначити переваги та недоліки порівняно з вже існуючими аналогічними системами. Найближчими конкурентами системи є Arlo Pro4 та Reolink Argus 2. Порівняємо дані системи з нашими, для визначення сильних та слабих сторін кожної з них (табл. 5.2).

Таблиця 5.2 – Визначення сильних, слабких та нейтральних сторін системи

№	Техніко-економічні характеристики ідеї	Продукція конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Моя система	Arlo Pro4	Reolink Argus 2			
1	Зручність використання системи	+	+	+		+	
2	Швидкість роботи системи	+	+	-		+	
3	Стиснення зображень	+	-	-			Стиснення розміру зображень для збереження більшої кількості даних
4	Оптимізація процесу передачі зображень	+	-	-			Оптимізація процесу передачі зображень для підвищення роботи системи
5	Мультиплатформа	+	-	-			Створення веб-додатку замість мобільного додатку, щоб користувачі усіх платформ мали доступ до системи

Із проведеного аналізу можна зробити висновок, що дана ідея має вагомі переваги порівняно з найближчими конкурентами.

5.2 Технічний аудит проекту

Далі нам необхідно провести аналіз альтернативних технологій, за допомогою яких можна реалізувати дану ідею стартап-проекту. Оскільки будь-який веб-додаток створюється за допомогою мов програмування HTML та CSS, або їх нащадків, то слід провести аналіз технологій для створення серверної частини

Таблиця 5.3 – Технології реалізації ідеї стартап-проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Розробка системи захисту приміщень з оптимізованою передачею стиснених зображень від робота до клієнта	Застосування мови програмування Node	Наявна	Вільна
2		Застосування мови програмування PHP		
3		Застосування мови програмування Python		

Зробивши аналіз доступних технологій, можна зробити висновок, що технологічна реалізація продукту можлива за допомогою використання декількох різних технологій. Для даної ідеї обрано технологію №1 з використанням та застосуванням мови програмування Node.JS.

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Далі необхідно провести аналіз впровадження та виведення даного стартап-проекту на ринок, визначити можливості та загрози в процесі запуску ідеї (табл. 5.4).

Таблиця 5.4 – Попередня характеристика потенційного ринку

№ п/п	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	1 (виробники відеокамер)
2	Загальний обсяг продаж, грн/ум.од	~100 000 грн/ум.од
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Наявність сховища у користувача
5	Специфічні вимоги до стандартизації та сертифікації	Отримання ліцензії від ЗОП (засоби охоронного призначення)
6	Середня норма рентабельності в галузі (або по ринку), %	100%

На основі проведеного аналізу, враховуючи кількість головних гравців по ринку, зростаючу динаміку ринку, мінімальну кількість конкурентів та високу8

норму рентабельності можна зробити висновок, що на даний момент, ринок для входження стартап-продукту є привабливим.

Далі потрібно визначити цільову аудиторію стартап-проекту (табл. 5.5).

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проєкту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Потреба в новому методі захисту приміщень та оселі	Цільовою аудиторією є підприємства із сучасним обладнанням і високим рівнем автоматизації виробництва та власники квартир або приватних осель	Клієнти прагнуть збільшити рівень захисту їх будівель та мати можливість спостерігати за приміщенням віддалено	Вимогами клієнтів є надійність та стабільність системи, зручність використання, висока швидкодія

На основі проведеного аналізу цільової аудиторії, а також порівняння зацікавленості між різними цільовими групами, можна зробити висновок, що дана система є однаково привабливою для підприємств та приватних власників.

Далі необхідно зробити аналіз ринкового середовища, а саме визначити фактори основних загроз, що можуть вплинути на продаж розробленої системи (табл. 5.6).

Таблиця 5.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Вихід конкурентів на ринок	Після виходу продукту на ринок можлива поява конкурентів	Нові інновації для приваблення нових клієнтів
2	Наявність перешкод	У приміщені може з’явитися перешкода, яка обмежить рух платформи	Створення алгоритму об’їзду перешкод
3	Знищення БД	Раптова помилка БД та втрата усіх даних	Періодичне резервування даних на хмарній платформі

Закінчення таблиці 5.6

4	Кібератака	Перевантаження серверів додатку, що унеможливить роботу	Завчасний аудит інформаційної безпеки, застосування резервних серверів

Далі необхідно розглянути основні фактори тих можливостей, які можуть вплинути на розроблений продукт під час виходу його на ринок (табл. 5.7).

Таблиця 5.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Новий ринок збуту	Розповсюджувати товар за межами країни	Локалізація продукту на мову країни-покупця та початок рекламної компанії
2	Співпраця з користувачем	Додавання нового функціоналу щодо потреб конкретного споживача	Відповідна зміна програмного продукту

Наступним кроком необхідно виконати ступеневий аналіз конкуренції (табл. 5.8).

Таблиця 5.8 - Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - чиста	Один продавець пропонує товар, який немає близьких замінників	Пропонує нові функції, що є точнішими та більш оптимізованими
2. За рівнем конкурентної боротьби - міжнаціональний	Ринок орієнтований на компанії зі всього світу	Локалізація продукту (переклад додатку на інші мови)
3. За галузевою ознакою - внутрішньогалузева	Додаток можна використовувати тільки для оптимізації фотографій	Розвинути функціонал додатку для покриття інших можливостей
4. Конкуренція за видами товарів: - товарно-видова	Усі методи виконують одну функцію – оптимізацію зображень	Можливість покращити якість зображення та зменшення її ваги
5. За характером конкурентних переваг - нецінова	Додаток використовується паралельно з системами відеоспостереження	Можливість інтеграції додатку до інших систем відеоспостереження

Закінчення таблиці 5.8

6. За інтенсивністю - не марочна	Існуючі методи оптимізації зображень нікому не належать (деяка сукупність методів може йти у одному пакеті)	Розвиток бренду для досягнення популярності по всьому світу

Далі необхідно виконати аналіз конкуренції за М. Портером (табл. 5.9).

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	Виробники відеокамер спостереження	Бар'єром входження в ринок є неготовність користувачів користуватися рухомими платформами для захисту	Постачальники мають вміти підключати платформу до існуючої БД	Споживачі повинні вміти користуватися функцією технічного сапорту	Відеокамери спостереження
Висновки:	Інтенсивність не є великою, оскільки конкуренти використовують кардинально різні підходи до рішення задачі	Можливість виходу існує: пристрій використовує принципово новий підхід Строки виходу на ринок: одразу після релізу готового продукту	Не диктують	Не диктують	Консерватизм споживачів (Небажання користувачів розбиратися з роботою пристрою)

Провівши аналіз отриманої інформації можна зробити висновок, що створена система є конкурентоспроможною, а також існує певна конкуренція на ринку, але розглянутий прямий конкурент має низку недоліків. Тому при запуску даного продукту необхідно розробляти з урахуванням усіх побажань користувачів, а також максимально обіграти явні недоліки аналогічних систем. Також проведено ступеневий аналіз конкуренції, визначено характеристики за якими вона проявляється, а також їх вплив на діяльність підприємства.

Наступним кроком необхідно обґрунтувати фактори конкурентоспроможності розробленої системи (табл. 5.10).

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Функціональність	У конкурентів немає можливості переміщувати камери у просторі
2	Розмір зображень	Розмір зображень є одним з основним факторів конкурентоспроможності, для стартапу він є особливим через те, що він є джерелом даних
3	Стабільність	Здатність пристрою працювати без збоїв протягом довгого періоду часу

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін системи

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з … (назва підприємства)						
			–3	–2	–1	0	+1	+2	+3
1	Функціональність	10		*					
2	Розмір зображень	17	*						
3	Стабільність	14				*			

На основі аналізу отриманих даних сформуємо SWOT аналіз (табл. 5.12).

Таблиця 5.12 – SWOT – аналіз стартап-проекту

Сильні сторони: – Висока якість зображень – Висока швидкість передавання даних – Переміщення за допомогою рухомою платформи	Слабкі сторони: – Вихід конкурентів на ринок – Неготовність користувачів користуватися рухомими платформами для захисту
Можливості: - Інтеграція додатку в стару систему відеоспостереження	Загрози: - Складний процес інтеграції з продуктом

За допомогою отриманого SWOT-аналізу необхідно створити альтернативи ринкової поведінки для того, що забезпечити запуск стартап-проекту на ринку, а також оптимальний час ринкової реалізації (табл. 5.13).

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Інтеграція додатку в стару систему відеоспостереження	Висока ймовірність (75%)	Середні строки реалізації (2 місяці)

Закінчення таблиці 5.13

2	Ефективна реклама продукту, що зацікавить клієнтів	Середня ймовірність (50%)	Середні строки реалізації (2 місяці)

За результатами проведеного аналізу можна зробити висновок, що необхідно обрати презентацію системи на відповідних публічних заходах для ринкового впровадження стартап-проекту.

5.4 Розробка ринкової стратегії стартап-проекту

Визначення стратегії охоплення ринку передує саме розробленню ринкової стратегії стартап-проекту.

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Застосування на підприємствах із сучасним обладнанням і високим рівнем автоматизації виробництва	Продукт є покращенням попередніх методі захисту, тому готовність прийняття повинна бути високою	Через недоліки попередніх методів захисту, попит на продукт повинен бути високим	Конкуренції не існує, інтенсивність дуже мала	Готовність підприємств та забудовників використовувати мій продукт
2	Застосування в приватних будинка чи квартирах				
Які цільові групи обрано: підприємства та забудовники для підвищення рівня безпеки					

Беручи до уваги той факт, що в ролі цільової групи визначені приватні будівлі та підприємства, то вирішено обрати стратегію масового маркетингу.

Таблиця 5.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	Ринок: підприємства та забудовники	Пропозиція нового типу захисту	Оптимальне співвідношення роботи алгоритмів і функціональності	Стратегія диференціації

Таблиця 5.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Так	Ні. Мій програмний продукт може працювати поряд з системами відеоспостереження	Не буде	Стратегія Лідера

Таблиця 5.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Функціональність, точність та швидкість роботи	Стратегія диференціації	Мінімізація розміру фотографій без особливої втрати якості	Зручність використання Висока швидкість передачі даних Підтримка товару (support)

Після проведення аналізу, вирішено обрати в якості стратегії розвитку – стратегію диференціації та стратегію заняття конкурентної ніші.

5.5 Розроблення маркетингової програми стартап-проєкту

Таблиця 5.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Новий тип захисту з новими можливостями	Велика швидкодія, економія ресурсів	Стиснення розміру зображень, оптимізація процесу передачі зображень, веб-додаток для підтримки мультиплатформи

Наступним кроком необхідно визначити опис рівнів моделі товару.

Таблиця 5.19 – Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="3">Охоронна система повинна бути впровадженою зі швидкістю інтернету 1 Гб/с, мати 1 Тб пам’яті БД, передавати 10 фотографій в одиницю часу, рухатись зі швидкістю 1 м/с, мати роздільну здатність 12 Мп, мати кут огляду 360°, дальність ІЧ-підсвічування 20м.</td></tr>
<tr><td rowspan="5">II. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Керованість
2. Об’єм інформації, що зберігається
3. Швидкість оптимізації</td><td>М
М
М</td><td>Тх
Тх
Тх</td></tr>
<tr><td colspan="3">Якість: Гарантована висока якість зображень</td></tr>
<tr><td colspan="3">Пакування: доступ до репозиторію та рухома платформа</td></tr>
<tr><td colspan="3">Марка: SSS – Smart Security System</td></tr>
<tr><td>III. Товар із підкріпленням</td><td colspan="3">Існує технічна підтримка товару, яку можна викликати для налагодження роботи, виправки несправностей</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: за рахунок новітніх ідей та узагальнення сучасних алгоритмів оптимізації зображень</td></tr>
</table>

Таблиця 5.20 – Визначення меж встановлення вартості

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	Відеокамери (мін. Ціна 3500 грн.)	Товарів-аналогів немає	Вільно високий	Продаж продукту: 1000$ Підписка на оновлення: 50$ в місяць Технічна підтримка: 250$ за виклик

Таблиця 5.21 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Клієнти (підприємства та забудовники), які прагнуть підвищити рівень захисту	Функція Billing Функція refund	Глибина 1: для окремих користувачів	Сервісна модель розповсюдження без альтернативних каналів збуту

Таблиця 5.22 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Нерегулярні продажі	Гаряча лінія, персональний кабінет	Функціональність, інноваційність	Мотивація клієнта використовувати даний продукт	With our innovative smart security system, you'll always be aware of what is going on in your house

На основі цього розроблено ринкову програму, яка складається із переліку ключових переваг концепції потенційного товару, опис його моделі, визначення питань ціно становлення а також питань формування системи збуту на основі концепції маркетингової комунікації.

Висновки до розділу

В даному розділі описано головну ідею стартап-проекту. Наступним кроком проведено технологічний аудит ідеї проекту, в результаті якого визначено, що головна ідею проекту є здійсненною. На основі отриманих результатів проведено аналіз ринкових можливостей запуску стартап-проекту. Далі вирішено питання із розробкою ринкової стратегії проекту, а також із розробкою маркетингової програми стартап-проекту. Тому проаналізувавши всі пройдені етапи можна зробити висновок, що даний проект може бути впроваджений на ринок, а також можна бути впевненим у зайнятті ним його відповідної ринкової ніші.

ВИСНОВКИ

В даній магістерській дисертації в першому розділі визначено область дослідження проекту, об’єкт та предмет дослідження. Проведено аналіз існуючих систем захисту приміщень та їх функціональних можливостей, визначено основні проблеми та переваги цих систем. Розглянуто дві системи: Arlo Pro4 та Reolink Argus 2 – на основі яких, поставлено декілька задач для вирішення в даній магістерській дисертації.

Наступним кроком проведено аналіз існуючих форматів зображень, визначено їх особливості. Обрано растровий формат, на основі якого, розглянуто його варіації та можливості кожного з форматів. Проведено аналіз ступенів стиснення зображень, розміри кінцевих файлів при різних рівнях стиснення та якість вихідного зображення. На основі цього обрано формат MJPEG для відео-трансляції та WebP для зберігання зображень.

В третьому розділі розглянуто та обрано технології для створення серверної та клієнтської частин, а також для процесу перетворення та стиснення зображень. Для стиснення та оптимізації процесу передачі зображень, серверної частини охоронної платформи з камерою та сховища обрано програмну платформу Node та менеджер пакунків NPM. В якості бази даних обрано MongoDB, щоб зберегти інформацію про користувачів та налаштування системи. Для клієнтської частини обрано мову програмування HTML, CSS та JavaScript, без використання будь-яких фреймворків. Передача відео-потоку здійснюється за допомогою використання WebSocket протоколу.

В наступному розділі описано компоненти системи, на яких розробляється система. Спроектовано архітектуру системи, встановлена взаємодія всіх модулів між собою, які забезпечують її повноцінну роботу в цілому. Розроблено структуру бази даних та створено кінцеві точки доступу для взаємодії серверів між собою та передачі інформації до клієнтів. Далі, розроблено серверну частину та здійснено перетворення фотографій у формат WebP, після чого перетворені зображення стискуються в залежності від налаштувань системи, а потім перетворюються в бінарний вигляд для покращення процесу передачі файлів. В кінці створено

клієнтську частину для представлення історії збережених файлів користувачам, а також для доступу до онлайн відео-трансляції.

Останнім кроком проведено маркетингове дослідження стартап-проекту, визначено цільову аудиторію користувачів та стратегію виведення системи на ринок.

ПЕРЕЛІК ПОСИЛАНЬ

1. Формати й алгоритми стиснення зображень [Електронний ресурс]: https://www.prepressure.com/library/compression-algorithm
2. Процес передачі декількох файлів одночасно [Електронний ресурс]: https://medium.com/typecode/a-strategy-for-handling-multiple-file-uploads-using-javascript-eb00a77e15f
3. WebSocket аутентифікація за допомогою JWT [Електронний ресурс]: http://iostreamer.me/ws/node.js/jwt/2016/05/08/websockets_authentication.html
4. Конвертація зображень в бінарний вигляд [Електронний ресурс]: https://www.tutorialspoint.com/how-to-convert-an-image-to-blob-using-javascript
5. Формати зображень [Електронний ресурс]: https://web.dev/choose-the-right-image-format/
6. Рівні стиснення зображень [Електронний ресурс]: https://web.dev/compress-images/
7. Використання WebP зображень [Електронний ресурс]: https://web.dev/serve-images-webp/
8. Програмна платформа Node.JS [Електронний ресурс]: https://uk.wikipedia.org/wiki/Node.js
9. Менеджер пакунків NPM [Електронний ресурс]: https://uk.wikipedia.org/wiki/Npm
10. Протокол WebSocket [Електронний ресурс]: https://uk.wikipedia.org/wiki/WebSocket
11. Система керування базами даних MongoDB [Електронний ресурс]: https://uk.wikipedia.org/wiki/MongoDB
12. Мова розмітки HTML [Електронний ресурс]: https://uk.wikipedia.org/wiki/HTML
13. Мова стилів CSS [Електронний ресурс]: https://uk.wikipedia.org/wiki/CSS

ДОДАТКИ

ДОДАТОК А

Результат перевірки на співпадіння

UNICHECK

User name:
Лісовиченко Олег Іванович

Check ID:
1005364706

Check date:
04.12.2020 14:26:47 EET

Check type:
Doc vs Internet + Library

Report date:
04.12.2020 14:28:28 EET

User ID:
76913

File name: **Колодій Олександр ІК-з91МП**

Page count: **45** Word count: **11020** Character count: **80217** File size: **110.33 KB** File ID: **1005657274**

5.28% Matches

Highest match: **2.57%** with Library source (File ID: **1000677550**)

1.52% Internet sources 26 ..... Page 47

4.29% Library sources 127 ..... Page 47

0% Quotes

Exclusion of quotes is off

Exclusion of references is off

0% Exclusions

No exclusions

ДОДАТОК Б

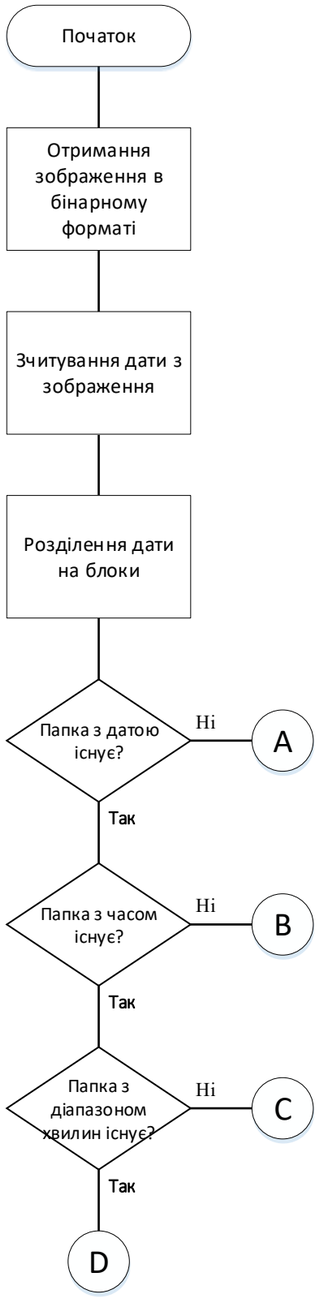
Плакати

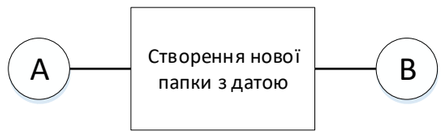
Архітектура системи

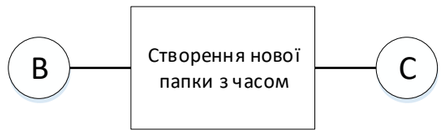
Демонстраційний плакат №1 «Архітектура системи»
до дипломної роботи на тему
«Оптимізація зображень роботизованої охоронної системи»

Виконав: студент гр. ІК-з91мп Колодій О.В.
Керівник: к. т. н., доцент Крилов Є.В.

Алгоритм отримання та збереження зображень



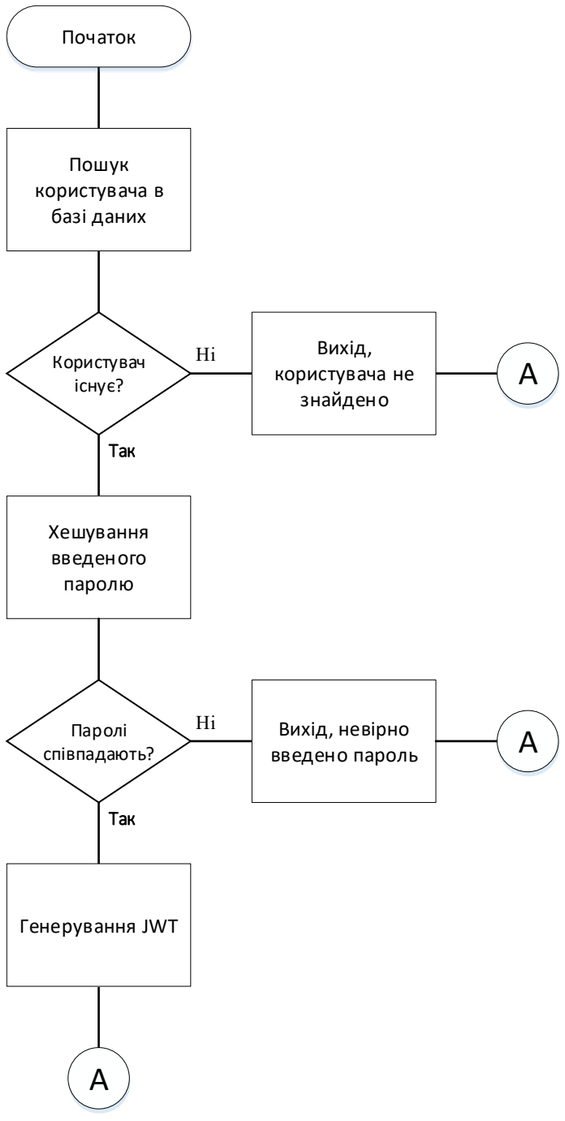




Демонстраційний плакат №2 «Алгоритм отримання та збереження зображень»
до дипломної роботи на тему
«Оптимізація зображень роботизованої охоронної системи»

Виконав: студент гр. ІК-з91мп Колодій О.В.
Керівник: к. т. н., доцент Крилов Є.В.

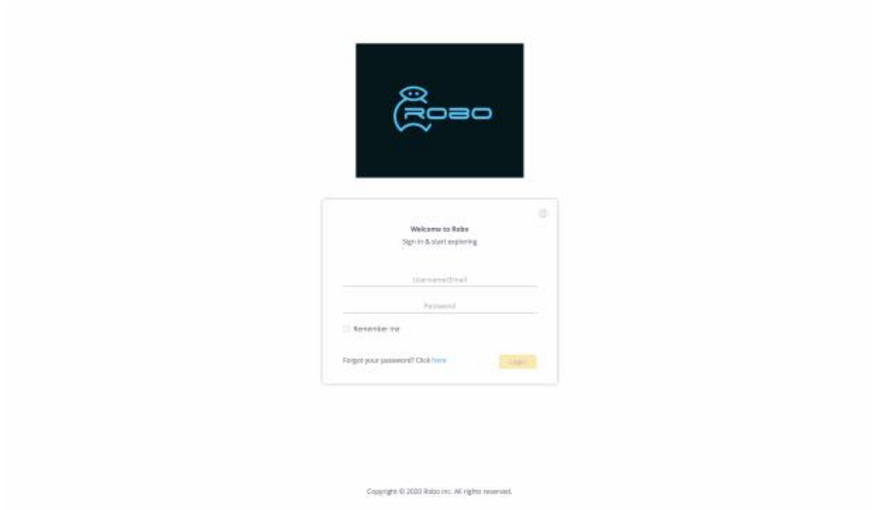
Алгоритм авторизації користувачів



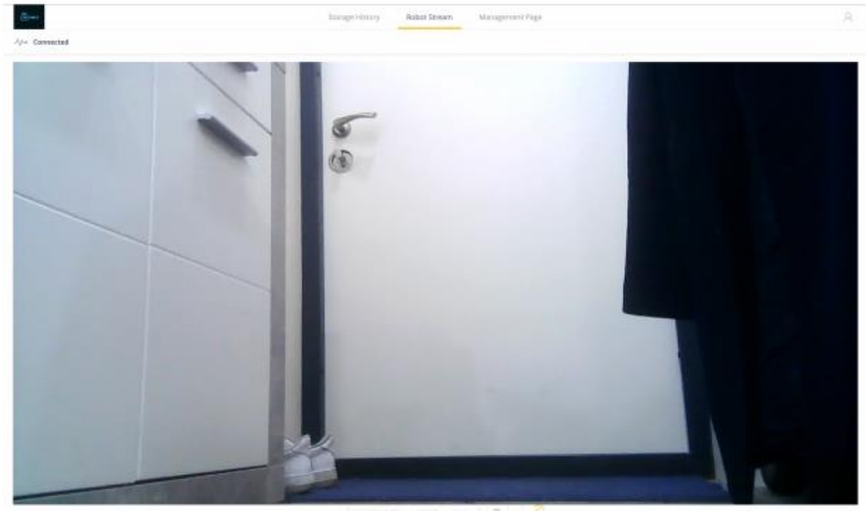
Демонстраційний плакат №3 «Алгоритм авторизації користувачів»
до дипломної роботи на тему
«Оптимізація зображень роботизованої охоронної системи»

Виконав: студент гр. ІК-з91мп Колодій О.В.
Керівник: к. т. н., доцент Крилов Є.В.

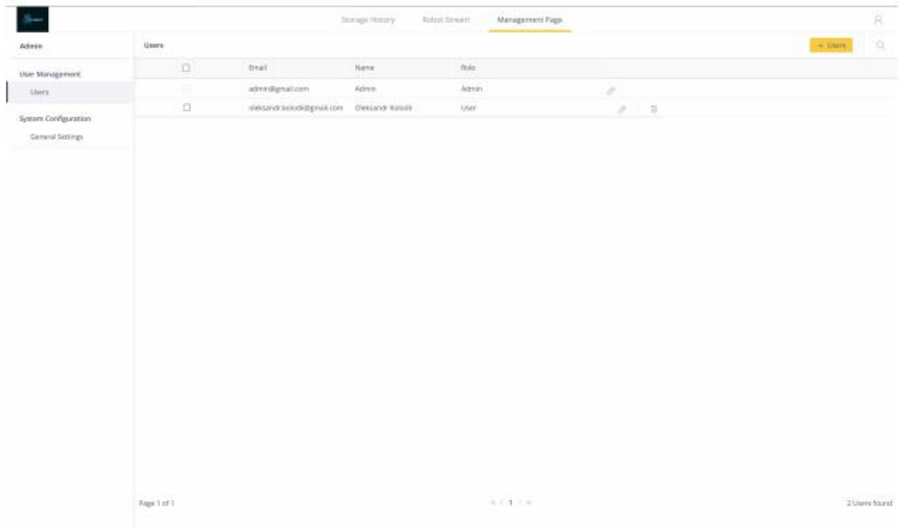
Інтерфейс користувача



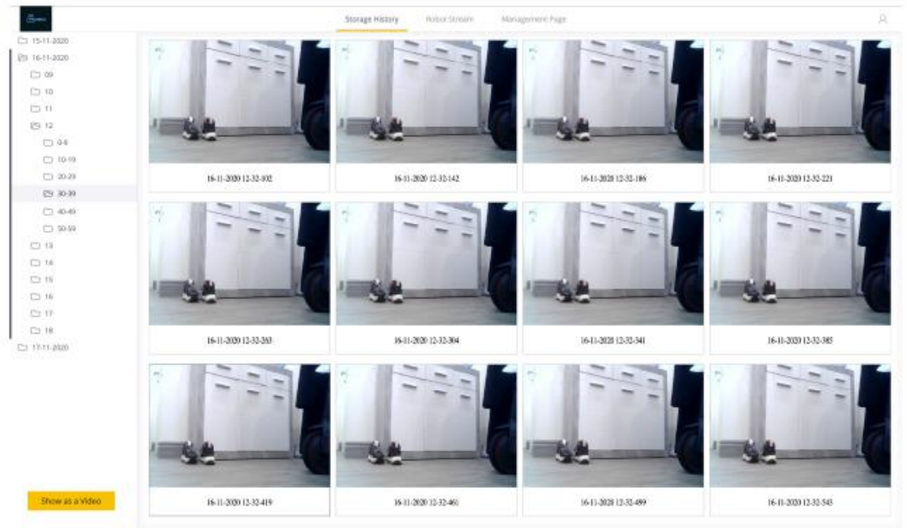
Сторінка авторизації



Сторінка відео-трансляції



Сторінка відео-трансляції



Text

Сторінка вмісту сховища

Демонстраційний плакат №4 «Інтерфейс користувача»
до дипломної роботи на тему
«Оптимізація зображень роботизованої охоронної системи»

Виконав: студент гр. ІК-з91мп Колодій О.В.
Керівник: к. т. н., доцент Крилов Є.В.

Аналіз ефективності стискання даних

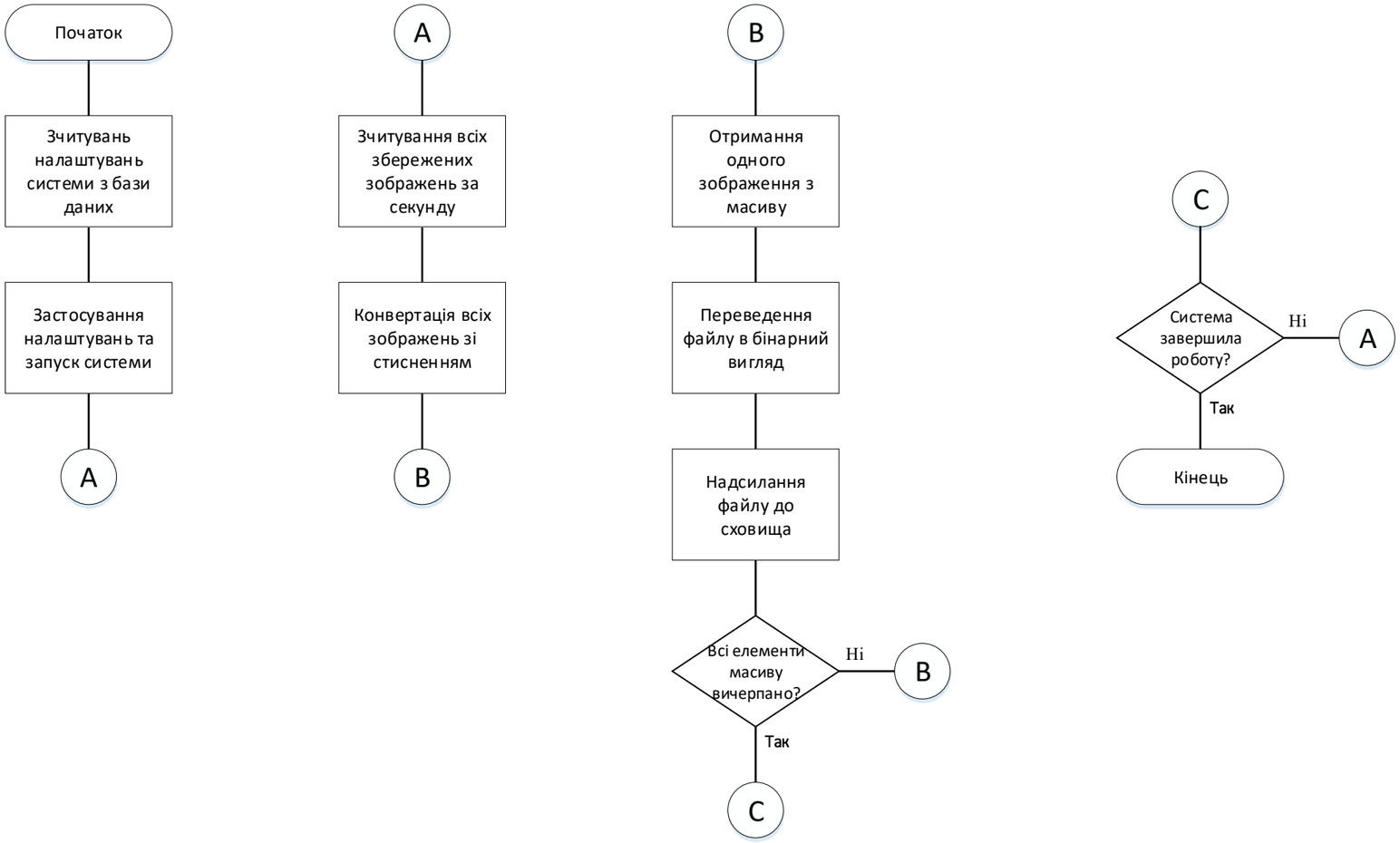
№ кадру	Base64	Binary
1	41.5 kB	27.5 kB
2	41.7 kb	27.9 kB
3	41.7 kb	27.7 kB
4	42.0 kB	28.0 kB
5	41.5 kB	27.7 kB
6	41.8 kB	27.8 kB
7	41.6 kB	27.6 kB

№ кадру	JPG	WebP (100%)	WebP (80%)	WebP (50%)
1	108.3 kB	90.2 kB	44.5 kB	29.2 kB
2	108.2 kB	90.2 kb	44.9 kB	29.5 kB
3	106.4 kB	89.3 kb	43.1 kB	29.1 kB
4	106.9 kB	89.6 kB	43.0 kB	29.9 kB
5	107.1 kB	93.8 kB	45.2 kB	30.8 kB
6	107.6 kB	91.0 kB	44.1 kB	30.0 kB
7	108.9 kB	90.5 kB	43.6 kB	29. 6 kB

Демонстраційний плакат №5 «Аналіз ефективності стискання даних»
до дипломної роботи на тему
«Оптимізація зображень роботизованої охоронної системи»

Виконав: студент гр. ІК-з91мп Колодій О.В.
Керівник: к. т. н., доцент Крилов Є.В.

Алгоритм оптимізації та надсилання зображень



Демонстраційний плакат №6 «Алгоритм оптимізації та надсилання зображень»
до дипломної роботи на тему
«Оптимізація зображень роботизованої охоронної системи»

Виконав: студент гр. ІК-з91мп Колодій О.В.
Керівник: к. т. н., доцент Крилов Є.В.